Portfolio Assignment 4, Methods 3, 2021, autumn semester

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

1.1 - Create a covariance matrix, find the eigenvectors and the eigenvalues (EH)

In [ ]:
```
# methods3 conda environment
# imports
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:
```
# common functions
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 [ ]:
```
data = np.load('megmag_data.npy')
pas_data = np.load('pas_vector.npy')
```

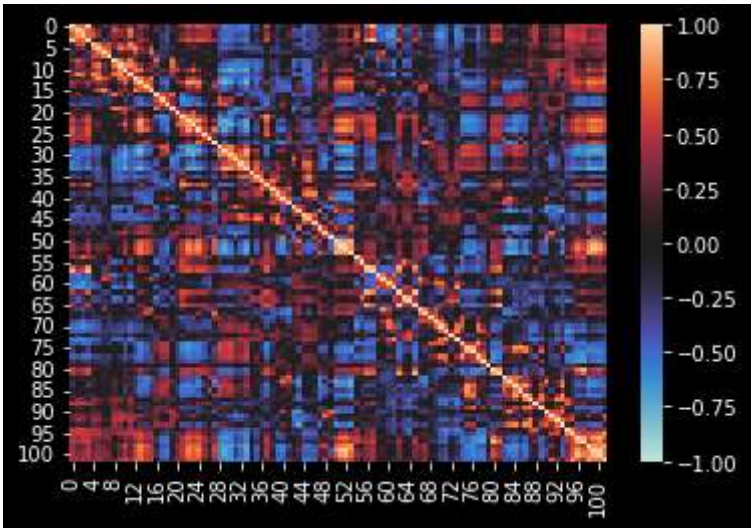
In [ ]:
```
# 1.1.ii) Equalize the number of targets in y and data using equalize_targets
data, y = equalize_targets(data, pas_data)
```

In [ ]:
```
# 1.1.iii) times
times=np.arange(-200, 804, 4)
# 1.1.iii) reduce to two dimensions only using 248ms
data_reduced = data[:, :, np.where(times == 248)[0][0]]
```

In [ ]:
```
# 1.1.iv) Scale the data using StandardScaler
```

```
sc = StandardScaler()
data_scaled = sc.fit_transform(data_reduced, y)
```

In [ ]:
```
# 1.1.v) Calculate the sample covariance matrix using np.cov
cov_matrix = np.cov(data_scaled, rowvar=False)
# plot using seaborn heatmap
sns.heatmap(cov_matrix, cmap='icefire', vmin=-1, vmax=1)
plt.show()
```



1.1.vi) What does the off-diagonal activation imply about the independence of the signals measured by the 102 sensors? (KV)

We see that the off-diagonal activation implies that the signals measured by the sensors are not independent. This makes sense as activity in the brain can involve multiple regions, and we would expect to see some correlation and covariance between sensors.

In [ ]:
```
# 1.1.vii Run np.linalg.matrix_rank on the covariance matrix - what integer value
rank = np.linalg.matrix_rank(cov_matrix)
print("Matrix rank: {}".format(rank))
```

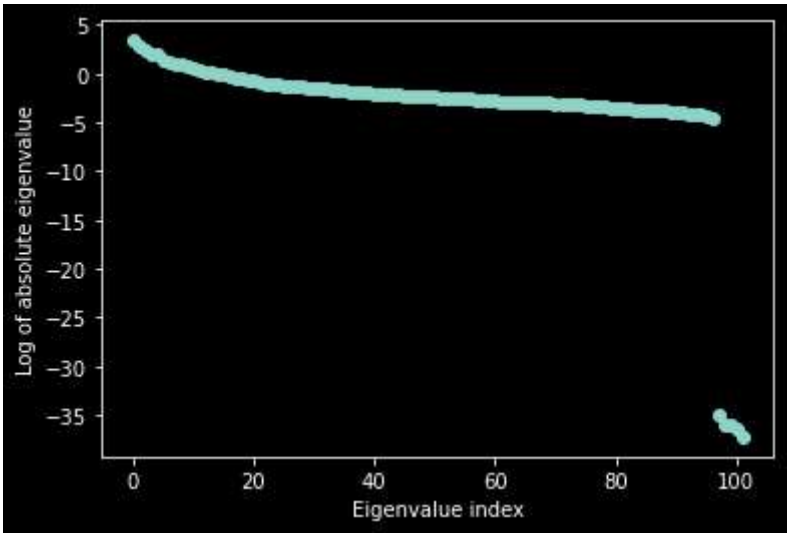
```
Matrix rank: 97
```

In [ ]:
```
# 1.1.viii) Find the eigenvalues and eigenvectors of the covariance matrix using r
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)
# Use np.real to retrieve only the real parts of the eigenvalues and eigenvectors
eigenvalues = np.abs(np.real(eigenvalues))
eigenvectors = np.real(eigenvectors)
```

1.2 - Create the weighting matrix W and the projected data, Z

In [ ]:
```
# 1.2.i to 1.2.iii) sort the eigenvectors and eigenvalues according to the absolut
# get the sorted indices of absolute values in reverse order
sorted_indices = np.argsort(eigenvalues)[::-1]
# sorted eigenvalues
eigenvalues = eigenvalues[sorted_indices]
# sorted eigenvectors
eigenvectors = eigenvectors[:, sorted_indices]
```

In [ ]:
```
# 1.2.iv) Plot the log of the eigenvalues
plt.plot(np.log(eigenvalues), 'o')
plt.xlabel('Eigenvalue index')
plt.ylabel('Log of absolute eigenvalue')
plt.show()
```



1.2.iv) are there some values that stand out from the rest? (LR)

The last 5 of the sorted log eigenvalues are much smaller than the previous 97. This matches the result of our matrix rank test which gives values above a threshold.

In [ ]:

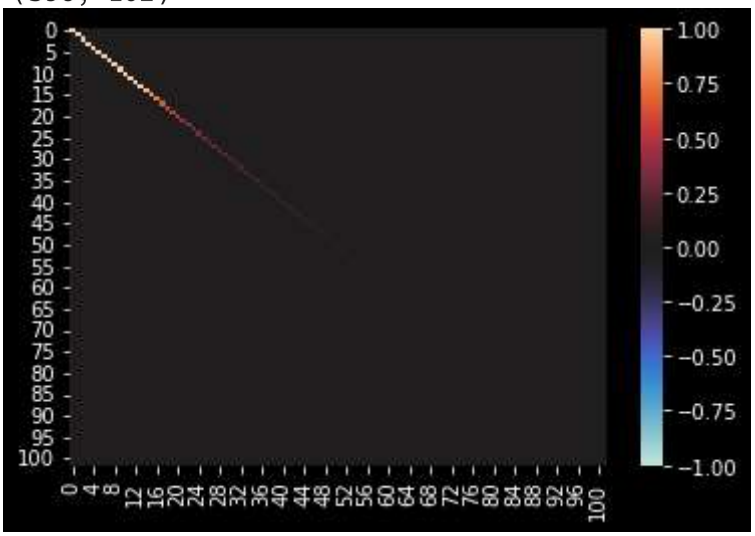
```
# 1.2.v - 1.2.vi) eigenvectors are the weighting matrix, create projected data Z =
Z = data_scaled @ eigenvectors
X = Z @ eigenvectors.T

# check if calculations are correct
closeness_check = np.isclose(data_scaled, X)
number_of_false_values = np.sum(closeness_check == False)
print("Number values that are not close: {}".format(number_of_false_values))
```

```
Number values that are not close: 0
```

In [ ]:

```
# 1.2.vii) Create a new covariance matrix of the principal components (n=102)
cov_matrix_pca = np.cov(Z, rowvar=False)
print(Z.shape)
# plot using seaborn heatmap
sns.heatmap(cov_matrix_pca, cmap='icefire', vmin=-1, vmax=1)
plt.show()
```

```
(396, 102)
```

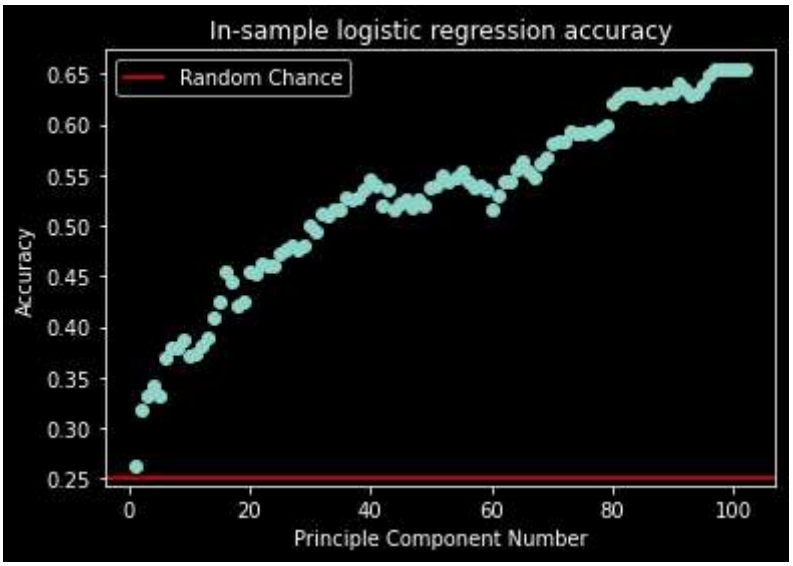


1.2.vii What has happened to the off-diagonals and why? (LR)

The weighting matrix is the orthogonal projection of the data onto the eigenvectors, this results in values that are not correlated with each other, giving a 0 value for the covariance matrix.

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

In [ ]:
```
# 2.1.i) run standard logistic regression (no regularization) based on Zdxk and y
lr = LogisticRegression(fit_intercept=False, solver='newton-cg')
# fit 102 models based on: k=[1,2,...,101,102] and d=102. For each fit get the cla
scores = list()
d = Z.shape[1]
for k in range(1, d+1):
    Z_kxd = Z[:, :k]
    lr.fit(Z_kxd, y)
    scores.append(lr.score(Z_kxd, y))

plt.scatter(np.arange(1, d+1), scores)
plt.axhline(0.25, color='red', label='Random Chance')
plt.xlabel('Principle Component Number')
plt.ylabel('Accuracy')
plt.title('In-sample logistic regression accuracy')
plt.legend()
plt.show()
```



2.1.ii) what is the general trend and why is this so? (EH)

Based on the above plot we could say the general trend is that addming more principle components results in a higher classification accuracy. This is because more variance is accounted for, but it also could mean that the model is overfitting.

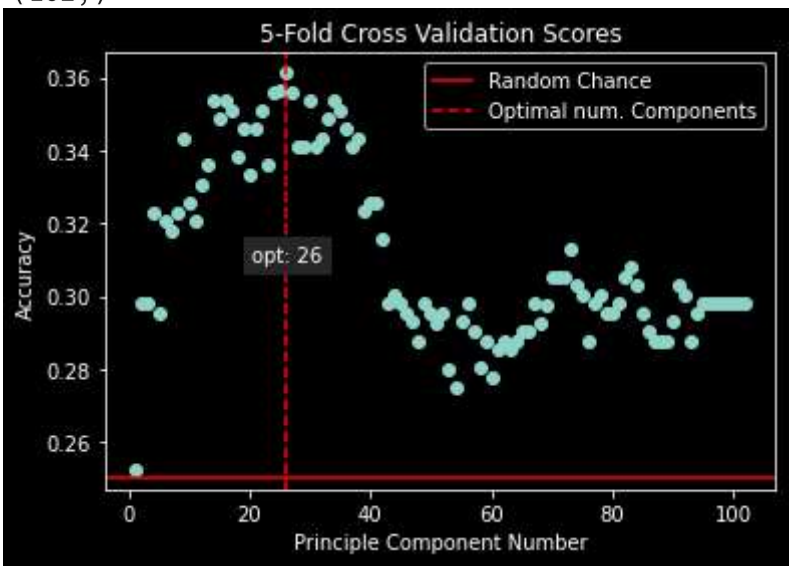
2.1.iii) In terms of classification accuracy, what is the effect of adding the five last components? Why do you think this is so? (KV)

The last 5 components have a negligible effect, this is because they are the principle components that have the least variance, adding them to our models doesn't provide much additional information.

In [ ]:
```
# 2.2) cross validate using cross_val_score and StratifiedKFold (LR)
cv = StratifiedKFold()
scores = []
for k in range(1, d+1):
  Z_k = Z[:, :k]
  scores.append(cross_val_score(lr, Z_k, y, cv=cv))

scores = np.asarray(scores)
# get mean score for each k
mean_scores = np.mean(scores, axis=1)
```

```
# get index of max
max_index = np.argmax(mean_scores)
grand_mean = np.mean(mean_scores)

print(mean_scores.shape)
plt.scatter(np.arange(1, d+1), mean_scores)
plt.axhline(0.25, color='red', label='Random Chance')
plt.axvline(max_index+1, color='red', label='Optimal num. Components', linestyle='
plt.text(max_index+1, grand_mean, 'opt: {}'.format(max_index+1), horizontalalignme
plt.xlabel('Principle Component Number')
plt.ylabel('Accuracy')
plt.title('5-Fold Cross Validation Scores')
plt.legend()
plt.show()
```

```
(102,)
```



2.2.ii) how is this plot different from the one in Exercise 2.1.ii? (VK)

This shows a different trend, rather than each additional component increasing the model accuracy, we see a peak at 26 components and after that a decline in accuracy which then levels out at around 70 components. Based on the first model we might assume that adding all the components would result in a higher accuracy, where as using cross-validation we see that the results of the previous model were likely due to overfitting.

2.2.iii) What is the number of principal components, $k_{max\_accuracy}$, that results in the greatest classification accuracy when cross-validated?

The peak model performance is 26 components, at around 36% accuracy.

In [ ]:

```
print("Maximum classification accuracy: {:.2%}, classifaction accuracy with full d
```

```
Maximum classification accuracy: 36.13%, classifaction accuracy with full dataset:
29.79%, improvement: 6.33%
```

2.2.iv) How many percentage points is the classification accuracy increased with relative to the to the full-dimensional, d, dataset? (EH)

using all of the components, the 5-fold cross-validation accuracy is increased by about 6.3%.

2.2.v) How do the analyses in Exercises 2.1 and 2.2 differ from one another? Make sure to comment on the differences in optimization criteria. (KV)

Both use a logistic regression model, the key difference here is that the first analysis is adding components and fitting the model to the entire dataset. 2.2 uses stratified 5-fold cross-validation which tries to ensure balanced classes across the folds, as well as training and testing on different subsets of the data. This is a more robust method of testing the model, as it is less prone to overfitting and while the classification accuracy may seem lower, the accuracy in 2.1 is likely overinflated.

In [ ]:

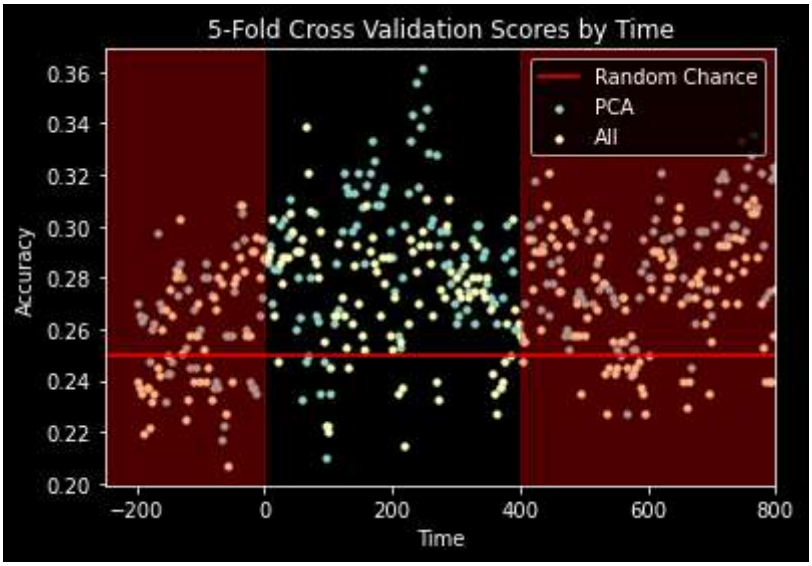
```
# 2.3.i) For each of the 251 time samples, use the same estimator and cross-valida
scores_pca = []
scores_all = []
time_steps = len(times)
print("Running analyses for {} time samples from {}ms to {}ms:".format(time_steps,
for t in range(time_steps):
  print("{}ms".format(times[t]), end="", flush=True)
  # get data for current timestamp
  data_t = data[:, :, t]
  data_t_scaled = sc.transform(data_t)
  # use pca to reduce to k_max
  pca_kmax = PCA(n_components=max_index+1)
  pcs = pca_kmax.fit_transform(data_t_scaled)
  # calculate cross validation scores for reduced data
  scores_pca.append(cross_val_score(lr, pcs, y, cv=cv))
  print(".", end="", flush=True)
  # calculate cross validation scores for all data
  scores_all.append(cross_val_score(lr, data_t_scaled, y, cv=cv))
  print(".", end="", flush=True)
print("done\nPreparing plot...", flush=True)
```

```
Running analyses for 251 time samples from -200ms to 800ms:
-200ms..-196ms..-192ms..-188ms..-184ms..-180ms..-176ms..-172ms..-168ms..-164ms..-1
60ms..-156ms..-152ms..-148ms..-144ms..-140ms..-136ms..-132ms..-128ms..-124ms..-120
ms..-116ms..-112ms..-108ms..-104ms..-100ms..-96ms..-92ms..-88ms..-84ms..-80ms..-76
ms..-72ms..-68ms..-64ms..-60ms..-56ms..-52ms..-48ms..-44ms..-40ms..-36ms..-32ms..-
28ms..-24ms..-20ms..-16ms..-12ms..-8ms..-4ms..0ms..4ms..8ms..12ms..16ms..20ms..24m
s..28ms..32ms..36ms..40ms..44ms..48ms..52ms..56ms..60ms..64ms..68ms..72ms..76ms..8
0ms..84ms..88ms..92ms..96ms..100ms..104ms..108ms..112ms..116ms..120ms..124ms..128m
s..132ms..136ms..140ms..144ms..148ms..152ms..156ms..160ms..164ms..168ms..172ms..17
6ms..180ms..184ms..188ms..192ms..196ms..200ms..204ms..208ms..212ms..216ms..220ms..
224ms..228ms..232ms..236ms..240ms..244ms..248ms..252ms..256ms..260ms..264ms..268m
s..272ms..276ms..280ms..284ms..288ms..292ms..296ms..300ms..304ms..308ms..312ms..31
6ms..320ms..324ms..328ms..332ms..336ms..340ms..344ms..348ms..352ms..356ms..360ms..
364ms..368ms..372ms..376ms..380ms..384ms..388ms..392ms..396ms..400ms..404ms..408m
s..412ms..416ms..420ms..424ms..428ms..432ms..436ms..440ms..444ms..448ms..452ms..45
6ms..460ms..464ms..468ms..472ms..476ms..480ms..484ms..488ms..492ms..496ms..500ms..
504ms..508ms..512ms..516ms..520ms..524ms..528ms..532ms..536ms..540ms..544ms..548m
s..552ms..556ms..560ms..564ms..568ms..572ms..576ms..580ms..584ms..588ms..592ms..59
6ms..600ms..604ms..608ms..612ms..616ms..620ms..624ms..628ms..632ms..636ms..640ms..
644ms..648ms..652ms..656ms..660ms..664ms..668ms..672ms..676ms..680ms..684ms..688m
s..692ms..696ms..700ms..704ms..708ms..712ms..716ms..720ms..724ms..728ms..732ms..73
6ms..740ms..744ms..748ms..752ms..756ms..760ms..764ms..768ms..772ms..776ms..780ms..
784ms..788ms..792ms..796ms..800ms..done
Preparing plot...
```

In [ ]:

```
# plot scores by time (VK)
mean_pca_scores = np.mean(scores_pca, axis=1)
mean_all_scores = np.mean(scores_all, axis=1)

fig, ax = plt.subplots()
ax.axhline(0.25, color='red', label='Random Chance')
ax.scatter(times, mean_pca_scores, label='PCA', s=10)
ax.scatter(times, mean_all_scores, label='All', s=10)
ax.axvspan(-250, 0, color="red", alpha=0.3)
ax.axvspan(400, 800, color="red", alpha=0.3)
```

```
plt.xlim([-250, 800])
plt.xlabel('Time')
plt.ylabel('Accuracy')
plt.title('5-Fold Cross Validation Scores by Time')
plt.legend(loc='upper right')
plt.show()
```



2.3.iii) Describe the differences between the two analyses - focus on the time interval between 0 ms and 400 ms - describe in your own words why the logistic regression performs better on the PCA-reduced dataset around the peak magnetic activity (EH)

The PCA reduced dataset performance tends to be higher than the full dataset, especially around peak activity. The full dataset contains the signal from all the sensors, but some of these may be less relevant to the response being investigated thereby introducing noise into the model. As hypothesized in 2.2.v, the overall accuracy of the full dataset was inflated due to overfitting which reduced its ability to be generalized to other time intervals. Unsurprisingly, as the PCA model underwent more robust validation, the accuracy at the previously modelled timestamp (248ms) remained the same (~36%), and seems to be able to generalize to time intervals around the peak activity, where the signal is likely similar.