

practical_exercise_10 , Methods 3, 2021, autumn semester

[FILL IN YOUR NAME]

[FILL IN THE DATE]

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
 - ii. Equalize the number of targets in `y` and `data` using `equalize_targets`
 - iii. Construct `times=np.arange(-200, 804, 4)` and find the index corresponding to 248 ms - then reduce the dimensionality of `data` from three to two dimensions by only choosing the time index corresponding to 248 ms (248 ms was where we found the maximal average response in Assignment 3)
 - iv. Scale the data using `StandardScaler`
 - v. Calculate the sample covariance matrix for the sensors (you can use `np.cov`) and plot it (either using `plt.imshow` or `sns.heatmap (import seaborn as sns)`)
 - vi. What does the off-diagonal activation imply about the independence of the signals measured by the 102 sensors?
 - vii. Run `np.linalg.matrix_rank` on the covariance matrix - what integer value do you get? (we'll use this later)
 - viii. Find the eigenvalues and eigenvectors of the covariance matrix using `np.linalg.eig` - note that some of the numbers returned are complex numbers, consisting of a real and an imaginary part

(they have a j next to them). We are going to ignore this by only looking at the real parts of the eigenvectors and -values. Use `np.real` to retrieve only the real parts

- 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).
 - ii. Then, we will find the correct ordering of the indices and create an array, e.g. `sorted_indices` that contains these indices. We want to sort the values from highest to lowest. For that, use `np.argsort`, which will find the indices that correspond to sorting the values from lowest to highest. Subsequently, use `np.flip`, which will reverse the order of the indices.
 - iii. Finally, create arrays of sorted eigenvalues and eigenvectors using the `sorted_indices` array just created. For the eigenvalues, it should like this `eigenvalues = eigenvalues[sorted_indices]` and for the eigenvectors: `eigenvectors = eigenvectors[:, sorted_indices]`
 - iv. Plot the log, `np.log`, of the eigenvalues, `plt.plot(np.log(eigenvalues), 'o')` - are there some values that stand out from the rest? In fact, 5 (noise) dimensions have already been projected out of the data - how does that relate to the matrix rank (Exercise 1.1.vii)
 - v. Create the weighting matrix, W (it is the sorted eigenvectors)
 - vi. Create the projected data, Z , $Z = XW$ - (you can check you did everything right by checking whether the X you get from $X = ZW^T$ is equal to your original X , `np.isclose` may be of help)
 - vii. Create a new covariance matrix of the principal components ($n=102$) - plot it! What has happened off-diagonal and why?

```
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
```

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 $Z_{n \times k}$ y (the target vector). Fit (`.fit`) 102 models based on: $k = [1, 2, \dots, 101, 102]$ and $d = 102$. For each fit get the

- classification accuracy, (`.score`), when applied to $Z_{d \times k}$ and $Z_{n \times k}$ and y . This is an in-sample validation. Use the solver `newton-cg` if the default solver doesn't converge
- ii. Make a plot with the number of principal components on the x -axis and classification accuracy on the y -axis - what is the general trend and why is this so?
 - iii. In terms of classification accuracy, what is the effect of adding the five last components? Why do you think this is so?
- 2) Now, we are going to use cross-validation - we are using `cross_val_score` and `StratifiedKFold` from `sklearn.model_selection`
- i. Define the variable: `cv = StratifiedKFold()` and run `cross_val_score` (remember to set the `cv` argument to your created `cv` variable). Use the same `estimator` in `cross_val_score` as in Exercise 2.1.i. Find the mean score over the 5 folds (the default of `StratifiedKFold`) for each k , $k = [1, 2, \dots, 101, 102]$
 - ii. Make a plot with the number of principal components on the x -axis and classification accuracy on the y -axis - how is this plot different from the one in Exercise 2.1.ii?
 - iii. What is the number of principal components, $k_{max_accuracy}$, that results in the greatest classification accuracy when cross-validated?
 - iv. How many percentage points is the classification accuracy increased with relative to the to the full-dimensional, d , dataset
 - 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.
- 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)
 - ii. Plot the classification accuracies for each time sample for the analysis with `PCA` and for the one without in the same plot. Have time (ms) on the x -axis and classification accuracy on the y -axis
 - 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