Binary classification of EEG data to overt behavior of motor inhibition

Marco Caspani (3181520)

Supervisors

Prof. Dr. J.L. Kenemans,

Prof. Dr. S. van der Stigchel

Utrecht University

Binary classification of EEG data to overt behavior of motor inhibition

Marco Caspani

July 2022

Abstract

The study of motor inhibition, that means stopping a muscle response which is almost taking place, is important for neuroscience. Knowing in advance when the brain is going to fail at this inhibition is also very helpful for applications meant to assist humans and demand attention to restore concentration. In this work, we tried to uncover the theta ground truth related to motor inhibition in EEG data (Kandiah, 2020), that is, the presence of bigger theta waves (4Hz-8Hz) in the EEG signal coming from the frontal area of the scalp before successful inhibition. Then we trained a classifier to predict whether the brain was going to fail or not at motor inhibition, starting with the EEG data of the one-second window before the motor inhibition process even takes place. The objective was to achieve a high accuracy while at the same time visualizing a clear focus on the frontal electrodes. We expected higher scores for the frontal electrodes after associating the elements of the weight vector used for classification to the corresponding electrodes. Discovery and subsequent removal of outliers uncovered the theta ground truth in the data. On the other hand, different workflows that we followed did not reach a classification accuracy higher than random guessing (50%) using leave one out cross-validation. Only training and testing on the same dataset reached an accuracy of 73%, though with no apparent theta ground truth. Modifications to the classifier from the previous work of Galama, 2021 led to the discovery of theta ground truth, even though the accuracy stayed at 50%.

1 Introduction

In everyday life, the brain has to continuously decide which actions to perform. For example, motor actions are prepared before their execution, that means that before there is a visible external behavior, brain activity is changing in order to initiate the action. Sometimes, though, there might be a sudden external stimulus, a danger or an unexpected auditory or visual element that catches the attention and redirects our priorities. Therefore, the brain has to interrupt the ongoing action preparation to adapt to the changed environment. In order to do this, in the first place, the brain has to successfully perceive the stimulus that tells to stop the action which was already planned (stop stimulus). For this, attention plays a key role and the brain has to be fast enough to process the incoming external input to successfully stop the action. If the stop stimulus arrives too late, or attention is directed elsewhere, the stopping process can fail. As a consequence, the action that the brain was preparing to do gets finally executed and becomes visible.

The ability of stopping actions that are about to be performed can be tested through the so-called stop task. In short, this experiment consists of presenting a go stimulus to the subject, indicating they have to perform an action. A small-time interval later (SOA - Stimulus Onset Asynchrony) a stopping stimulus is presented, meaning that the subject has to stop the action they were about to do. By modulating the SOA, the subject accuracy for successful stops may change. In fact, if the SOA is long enough, the subject will already be performing the action, making the stop impossible.

Previous work, like O'Connell et al., 2009, showed that it is possible to predict human errors in attentional lapses from brain activity, even up to 20 seconds before they actually happened. For this reason, it becomes interesting and important to develop classifiers, mathematical models that analyze the brain activity and yield a binary prediction, success if there is not going to be a human error, fail otherwise. Methods for predictions can come from, for example, Kandiah, 2020 where they showed that significant frontal theta power appears in successful stops. This significant result should surely appear in a classifier of importance because it reflects a main effect observable in the inhibition process. For this reason, one factor for which the prediction is made possible is looking at theta power, because in the successful condition theta power is greater. A classifier that only looks at this variable should be able to classify a success vs a fail better than chance (i.e. 50% accuracy).

The study of this prediction is important for both theoretical and practical applications. In the field of Neuroscience, studying how to improve classifiers that predict errors in action inhibition can lead to contribution to how attentional processes work. On the other hand, for practical applications, it may be useful to predict if a driver is going to have an attentional lapse to notify them before it is too late. A relevant stimulus could potentially be harmful if not properly attended, for example a child who suddenly crosses the street. Notifying the driver would mean making sure that the driver inhibits the action of holding the accelerator, to proceed with braking.

This knowledge over theta waves from literature tells that a good classifier should manifest, through its interpretation, the theta ground truth from Kandiah, 2020. In other words, interpretation means that the classifier focuses on the theta range of the frontal electrodes to make its prediction, for example, by giving more importance to the weights associated to frontal theta power in the mathematical model of the classifier. And this led to our research question: is it possible to optimize the LDA classifier so that its interpretation also shows the theta ground truth?. And to a greater extent: what other insights would such a classifier give on the inhibition brain process? This research was aimed at improving the previous work made in Galama, 2021. In their work, they analyzed a preprocessed EEG data window of one second preceding the go-stimulus followed by the onset of a stopping stimulus, and tried to predict if the brain would successfully stop the ongoing action preparation. However, the accuracy of the linear discriminant analysis (LDA) classifier, which was used, revolved around 50%. An improvement of this classifier was aimed at increasing the accuracy to have a classifier which is better than random chance, while at the same time integrating the theta ground truth that is expected to be in the EEG data.

2 Methods

In this section, we will outline the methods that we used in order to answer the research question. The first methods involve the validation that the data actually contain the expected theta ground truth. Then, since the old classifier code was not flexible enough to handle rapid changes to see the effect of modifications, we programmed a new Linear Discriminant Analysis (LDA) classifier. We proceeded with the improvement of the classifier by training it on different search spaces. We tried to discover the theta ground truth present in the data also from the classifier. For example, by doing the training on datasets where missing values from outliers were filled, or by trying to remove as little information as possible while removing the outliers.

2.1 Data

The used dataset is EEG data from 29 participants, acquired from Kenemans et al., 2022. The participants (age: M = 22.8, SD = 3.26), students at Utrecht University, were all healthy, with normal vision and normal hearing. Previous informed consent, approved by the ethics

	Fp1	AF7	AF3	F1	F3	F5	F7	FT7	FC5	FC3	...	P6	P8	P10	PO8	PO4	O2	class	freqc	trial	ptp
4	0.539	0.436	0.524	0.575	0.433	0.141	0.909	0.341	0.888	0.518	...	0.987	0.477	2.338	0.990	0.069	1.144	f	4	0.0	001
5	1.337	1.619	0.396	0.406	0.762	0.656	0.864	0.882	1.020	0.506	...	1.335	0.376	0.042	0.164	0.069	0.377	f	5	0.0	001
6	0.360	1.997	0.510	0.747	0.361	0.971	1.644	0.723	0.246	0.130	...	0.079	0.093	0.019	0.126	0.005	0.324	f	6	0.0	001
7	3.318	0.387	3.576	3.225	2.748	0.678	1.048	0.568	1.598	2.703	...	0.169	0.468	0.426	0.013	0.099	0.173	f	7	0.0	001
36	2.997	7.029	3.688	1.068	1.434	2.835	2.394	0.331	0.484	0.576	...	0.063	0.516	2.575	0.752	0.018	0.979	f	4	1.0	001
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9191	11.544	9.744	14.468	15.362	13.128	7.919	4.777	3.457	4.521	11.296	...	1.457	2.182	5.168	3.248	1.010	4.641	s	7	181.0	112
9220	15.149	13.635	13.522	15.373	11.261	9.293	13.113	8.539	5.620	4.494	...	7.576	0.779	1.206	2.082	0.980	5.977	s	4	182.0	112
9221	1.574	3.086	0.982	2.964	2.181	1.196	1.958	2.239	0.660	2.479	...	0.696	4.310	5.392	1.940	1.111	3.781	s	5	182.0	112
9222	27.735	12.515	26.768	23.780	12.442	8.910	9.469	8.478	4.603	6.994	...	3.589	8.143	7.202	2.705	0.121	1.403	s	6	182.0	112
9223	3.738	1.130	5.714	5.867	2.605	1.473	0.722	0.360	0.209	0.329	...	1.101	1.008	1.672	0.810	1.040	0.469	s	7	182.0	112

25224 rows × 68 columns

Figure 1: EEG data after preprocessing. Each row of the table is a data point where there are power values for each electrode. Furthermore, in each row there is the class of the data point, fail (f) or success (s), that says if in that trial inhibition was successful or not; the frequency range (freqc) that says which frequency range the power values refer to, for example, $freqc = 4$ means the values are the power values of the frequencies from 4Hz to 5Hz; the trial column indicates the trial number to which the data point belongs to; the ptp column indicates the number of the participant of the row data.

committee of the University Medical Centre Utrecht, was signed. Employing the ActiveTwo Biosemi system, 64 Ag-AgCl electrodes were used, placed following the 10/10 system, and the signal was sampled at 2048Hz. The signal was online referenced to the Common Mode Sense/Driven Right Leg electrode and low pass filtered at DC to 400 Hz. The brain activity was recorded during the SST, described in the introduction, where the participant's stop-accuracy was titrated to approximately 50% (Galama, 2021). From the three different paradigms in Kenemans et al., 2022 in this work we will use the data from the auditory SST. This means the experiment where the go stimulus is visual, and it is followed by an auditory stop stimulus.

For the preprocessing phase, the same steps used in Galama, 2021 were used. We report here, until the end of this paragraph, the preprocessing section in Galama, 2021, from page 4 to 5, which summarizes well the preprocessing steps: BrainVision Analyzer 2.1 was used for preprocessing. Spline interpolation has been applied, but no channels were interpolated. The EXG5 channel was used for re-referencing all the other channels. A low cutoff at 0.5 Hz and a high cutoff at 28.8 Hz with zero phase shift Butterworth filters of order 2 were applied. The sampling rate was reduced to 64 Hz. There is no reason to presume that such a downsampling after including a low-pass filter at 28.8 Hz will affect the frequencies up until 20 Hz. The first segmentation had a segment position relative to go-stimuli reference markers of -1000.00 ms to 1500.00 ms, in which overlapping segments were allowed. Eye blinks were removed with the Gratton and Coles method. No other artifact rejection has been done, since this may cause unnecessary data loss, after which the remaining data may still be containing artifacts. Also, multivariate pattern decoding (and specifically linear discriminant analysis) is assumedly more robust to artefacts than the established or traditional preprocessing methods used in univariate ERP/ERF research (Carlson et al., 2019). The second segmentation selected all the 1000 ms intervals preceding go-stimuli followed by an (auditory) stop-signal and separated these (for failed and successful inhibition). A fast fourier transformation using the half-spectrum with a resolution of 1 Hz and a periodic Hanning data window of 10% length (with variance correction) resulted in non-complex power values, for every second before the go-stimulus was presented, divided into the two classes. How the data table looked like at this point can be seen in figure 1.

EEG data was also further preprocessed before computing the LDA classifier. The number of fail data points was balanced to the number of success data points. Note that one data point is one row of the dataframe in figure 1. In this way, guessing one prediction at random would

give an accuracy of 50%. The classifier could not try to predict the majority class, whether fail or success, to reach an accuracy better than 50%.

2.2 Uncovering the theta ground truth in the data

To validate that the data contain the theta ground truth, we had to check that in the success condition there was a higher theta power with respect to the fail condition. First, the preprocessed data were imported and outliers were removed. The outliers' removal consisted of the following: each data point, a single row of the entire data table, was removed if at least one component of the row vector was greater than five.

With the cleaned data (no outliers), the data of each participant were grouped by frequency range and class, fail or success, and averaged. This process was made in preparation of a repeated measures ANOVA (RM ANOVA) test to verify that the average values of frontal electrodes in the theta range (4-8Hz) were significantly different, validating in this way the presence of the theta ground truth in the data.

The RM ANOVA test was conducted, to verify a significant difference between the average of frequency values in the theta range, in the frontal area. For this purpose, the electrode FCz was used as representative of the frontal area. The test factors were two: "failsucc", for the fail and success conditions, and "freqc" (frequency class), for each frequency range. The levels of this latter were 4, 5, 6, 7 in order to run the ANOVA test on the theta range. For example, when the frequency range variable (freqc) is set to 4, this means considering the power value for the frequency range 4Hz-5Hz. Likewise for $freqc\ 2\ f5, 6, 7g$.

2.3 Linear Discriminant Analysis

The classifier we used for classification was the Linear Discriminant Analysis, or LDA, classifier. In a multidimensional space, the LDA finds a projection vector w onto which the projections of the data points are maximally separated. Projecting the w vector to the mean vector of the two classes, gives a scalar criterion c. In this way, if any data projection is smaller than c the data point is classified as class A. Likewise, if any projection is bigger than c the data point is classified as class B.

In our case, every data point was a vector of 64 dimensions, using all the available electrodes from the 10/10 system, or only 26 electrodes (Fp1 AF7 AF3 F1 F3 F5 F7 FT7 FC5 FC3 FC1 Fpz Fp2 AF8 AF4 AFz Fz F2 F4 F6 F8 FCz FC2 FC4 FC6 FT8) if considering the frontal electrodes only. As a consequence, the weight vector would assume a dimensionality that mirrors the number of electrodes considered as input features, while the criterion c always remains a scalar.

The procedure to compute the weight vector w and the scalar criterion c, also referred to as training of the classifier, was taken from Galama, 2021:

- Calculate the mean vector of data points labelled as class f, and the mean vector of data points labelled as class s. This process is repeated for each participant. In the formulas, X_i refers to the vector of power values for all electrodes, while the sum of these vectors is to be intended as the pointwise sum of their components. n_f and n_s are respectively the number of data points of the fail class and the number of data points for the success class.

$$M_f = \frac{1}{n_f} \sum_{i, Xi2C_f} Xi; \qquad M_s = \frac{1}{n_s} \sum_{i, Xi2C_s} Xi \tag{1}$$

- Calculate the two scatter matrices separately for data of the fail and success classes. Scatter matrices are estimations of the covariance matrices when it is not possible to calculate the

covariance, or it is too computationally expensive to do so. Because this was not the case, we interchanged the computation to covariance matrices calculations

$$S_f = covariance\_mtx(Data_f); \qquad S_s = covariance\_mtx(Data_s); \tag{2}$$

- Find the within class variance matrix by adding the two matrices S_f and S_s calculated in the previous step

$$S_w = S_f + S_s \tag{3}$$

- In order to find the weight vector we have to multiply the inverse of S_w with the difference of the two mean vectors found in the first step.

$$w = S_w{}^{1}(M_f \quad M_s) \tag{4}$$

- Finally, we can find the discriminant criterion c by projecting the weight vector onto the average of the two mean vectors M_f and M_s

$$c = w \quad ((M_f + M_s)/2) \tag{5}$$

- To classify a data point d, row vector of the data table (see for example figure 1), project the weight vector on the data point by taking the dot product and then compare it to the criterion c.
  $w \quad d < c$) classify as success
  $w \quad d > c$) classify as fail
  From equation 4, we can see this is the correct way for classification, and not the other way around because w is the normalization of the difference between the two mean vectors M_f and M_s, not between M_s and M_f.

2.4 Uncovering the theta ground truth trough LDA

Once we knew that in the data there was the theta ground truth, thanks to the strategy we described in section 2.2, we tried to find the same through the LDA classifier. In the following procedures we imported data, referring to the full data, with all the 64 electrodes of the 10/10 system. We only imported data of the frequency range 4Hz - 8Hz, since this is the range where the theta ground truth lies. Note that every procedure starts then with the removal of outliers because in this way we are sure to have the theta ground truth in the data. The last step before the computation of the classifier is to balance the number of fail and success data points. The balancing process works in the following way: when it was used, if in the dataframe taken into consideration there were more fail data points (rows) than success ones, or vice versa, we sampled random rows from the majority set, fail or success, to match the cardinality of the minority set, success or fail. For instance, let n be fail data points and m success data points. If $n > m$, we kept m data points from the fail class. Vice versa, if $n < m$, we kept n data points from the success class.

In this way, there would not be a majority class, and the classifier could not reach an accuracy better than chance (50%) just by guessing a majority class, because it does not exist after balancing. A baseline classifier would then be the random classifier, which would have an accuracy of 50% since this problem is a binary classification. If our LDA were to perform better than 50% then it would mean that it worked better than chance.

To test the significance of the accuracies, we compared them against the table containing the sample-size specific accuracy thresholds for significance, as a function of arbitrary p-values restrictions. Table 1 contains these thresholds.

Below, we report all LDA flows we followed. The source code of the following workflows is attached to appendix A.

significant accuracies thresholds for 2-classes

n	p<0.05	p<0.01	p<10^{-3}	p<10^{-4}
20	70.0%	75.0%	85.0%	90.0%
40	62.5%	67.5%	75.0%	77.5%
60	60.0%	65.0%	70.0%	73.3%
80	58.7%	62.5%	67.5%	70.0%
100	58.0%	62.0%	65.0%	68.0%
200	56.0%	58.0%	61.0%	63.0%
300	54.7%	56.7%	59.0%	60.7%
400	54.0%	55.7%	57.7%	59.2%
500	53.6%	55.2%	57.0%	58.2%

Table 1: Accuracy thresholds for desired significance, given the number of data points in the sample of a population (Combrisson & Jerbi, 2015). In the first column, there is n: the number of data points in the sample. On the other headers, there is the arbitrary significance which is wanted.

- In the first workflow, we did not apply the Leave One Out (LOO) procedure, because the LDA classifier was trained on the totality of data within each participant. The workflow was the following:
  - Import the dataframe of all data
  - Remove all the rows with at least one outlier (value > 5)
  - For each participant ptp:
    - consider only the data of ptp
    - balance the number of fail and success rows
    - *compute* the LDA classifier
    - *classify* all data of ptp using the weight vector $\mathbf{w}$ and the criterion $\mathbf{c}$
    - average accuracies of all predictions and go to the next participant
- In the second workflow, we removed all rows containing an outlier from the dataframe and then applied the LOO. The workflow was the following:
  - Import the dataframe of all data
  - Remove all the rows with at least one outlier (value > 5)
  - For each participant ptp:
    - consider only the data of ptp
    - balance the number of fail and success rows
    - For each row in the dataframe of ptp:
      - leave row out
      - *compute* the LDA classifier on the rest of the rows
      - *classify* the left out row of ptp using the weight vector $\mathbf{w}$ and the criterion $\mathbf{c}$
    - average accuracies of all predictions and go to the next participant
- In the third workflow, we removed all cells containing an outlier from the dataframe, filled them with the mean of the column values and then applied the LOO. This means that for each cell of the dataframe table, if the cell value was greater than five, we removed the cell content, leaving a void. This void was later filled with the average of the other values, less than five, of the same column of the dataframe. For example, if the dataframe were to be the following

  the content of the cell at row 2, column 2 (7.029) would be an outlier because $7.029 > 5$, therefore this value it is replaced with the average of the non outlier values of the same column. Eventually, 7.029 is replaced with $(2.000 + 3.000)/2 = 2.500$

Fp1	**AF7**	**AF3**	**F1**	**...**	**PO8**	**PO4**	**O2**	**class**	**freqc**	**trial**	**ptp**
2.997	2.000	3.688	1.068	...	0.752	0.018	0.979	f	4	1.0	001
3.78	**7.029**	3.16	4.068	...	1.535	4.34	4.245	f	5	1.0	001
4.214	3.000	3.243	1.468	...	0.130	2.018	4.479	f	6	1.0	001

The workflow was the following:

- Import the dataframe of all data
- Remove all cells with a value greater than five
- For each participant ptp:
  - consider only the data of ptp
  - fill the missing cell values with the *mean* of the other column values
  - balance the number of fail and success rows
  - For each row in the dataframe of ptp:
    - leave row out
    - *compute* the LDA classifier on the rest of the rows
    - *classify* the left out row of ptp using the weight vector **w** and the criterion **c**
  - average accuracies of all predictions and go to the next participant

- In the fourth workflow, we removed all rows containing an outlier from the dataframe and then applied the LOO on the electrode x frequency search space obtained from the cleaned data. In this way, the search space contained 64 x 4 = 256 variables (64 electrodes, 4 frequency ranges: 4-5Hz, 5-6Hz, 6-7Hz, 7-8Hz). The operation in order to achieve this objective is called pivoting (https://pandas.pydata.org/docs/reference/api/pandas.pivot_table.html). Pivoting a dataframe means reshaping it in order to bring the frequency range information into the column names. After the pivoting, the dataframe columns will be named in the form "electrode_freqc" (e.g. "FCz_4", that is the column containing the power values of the FCz electrode for the frequency range 4Hz-5Hz). The workflow was the following:
  - Import the dataframe of all data
  - Remove all the rows with at least one outlier (value > 5)
  - For each participant ptp:
    - consider only the data of ptp
    - pivot the ptp dataframe on the freqc column (frequency range) in order to create an electrode x frequency search space.
    - fill the missing cell values with the *mean* of the other column values
    - balance the number of fail and success rows
    - For each row in the dataframe of ptp:
      - leave row out
      - *compute* the LDA classifier on the rest of the rows
      - *classify* the left out row of ptp using the weight vector **w** and the criterion **c**
    - average accuracies of all predictions and go to the next participant

- In the fifth workflow, we removed all cells containing an outlier from the dataframe, filled them with the mean of the other column values and then applied the LOO on the electrode x frequency search space obtained from the cleaned data.
  - Import the dataframe of all data
  - Remove all cells with a value greater than five
  - For each participant ptp:
    - consider only the data of ptp
    - pivot the ptp dataframe on the freqc column (frequency range) in order to create an electrode x frequency search space

        - fill the missing cell values with the *mean* of the other column values
        - balance the number of fail and success rows
        - For each row in the dataframe of ptp:
            - leave row out
            - *compute* the LDA classifier on the rest of the rows
            - *classify* the left out row of ptp using the weight vector **w** and the criterion **c**
        - average accuracies of all predictions and go to the next participant

- In the sixth workflow, we only considered data of the FCz electrode. We removed all rows containing an outlier from the dataframe and then applied the LOO.
    - Import the dataframe of FCz data of all participants
    - Remove all rows with a value greater than five
    - For each participant ptp:
        - consider only the data of ptp
        - balance the number of fail and success rows
        - For each row in the dataframe of ptp:
            - leave row out
            - *compute* the LDA classifier on the rest of the rows
            - *classify* the left out row of ptp using the weight vector **w** and the criterion **c**
        - average accuracies of all predictions and go to the next participant

After we tried the just described workflows, and noticed non-significant results, we also ran the same structures of workflows 2 - 5, but with different data in input. This time, we only considered the data of frontal electrodes (Fp1 AF7 AF3 F1 F3 F5 F7 FT7 FC5 FC3 FC1 Fpz Fp2 AF8 AF4 AFz Fz F2 F4 F6 F8 FCz FC2 FC4 FC6 FT8), again importing only data of the frequency range 4Hz - 8Hz, and kept the rest of the workflow the same. The reason for this strategy was to expect the classifier to handle less noise, since the theta ground truth deals with frontal electrodes only, thus achieving better accuracies.

2.5 Modifications to the LDA and workflows to uncover the theta ground truth

After we tested the workflows described in the previous section, yielding non-satisfactory results, we decided to apply some modifications to the computation of the classifier, or the workflows, in order to uncover the theta ground truth (i.e. greater score for frontal electrodes in the LDA weight vectors). Workflow 1 was the only one without the Leave One Out procedure (LOO), and it reached a high accuracy. At this point, we also wanted that other workflows, like workflow 2, achieved a high accuracy when the classifier is tested on unseen data, left out during the LOO.

We begin by reporting the modifications that did not lead to an improvement of workflow 2, applied singularly. We do not include detailed results in the results section because of time constraints, and especially because the accuracies of workflow 2 with the following modifications do not move from accuracies equal to 50%.

We applied the Principal Component Analysis (PCA) (Jolliffe & Cadima, 2016) to the preprocessed data without outliers. The PCA was a choice to decrease the dimensionality of the data, while preserving the variance, hopefully decreasing noise by not considering the principal components which supposedly reflect noise, because they present the least explained variance. We increasingly fed the principal component scores for $n = 2, 3, 4...10$ components to the LDA classifier, instead of the preprocessed data without outliers. The accuracy of workflow 2 after these steps was still 50%.

Another modification was made to the LOO procedure. We adjusted the trade-off between the left-out trials and trials for the weight vector estimation. This adjustment let us generate a classifier which is more generic given the fewer data for weight vector estimation, and at the

same time, with more averaging on the left-out data predictions, possibly yielding a higher accuracy for workflow 2. The leave out size was specifically increased to 10%, meaning that not only one data point was left out at each LOO iteration, but 10% of data points were left out and not included in the computation of the LDA weight vector. On the other hand, the classifier was tested on 10% of the data points, instead of 1 point only, at each iteration. We also tried with other leave out sizes, for example 5% and 20%, but in all cases the accuracy of workflow 2 remained at 50%, still not better than random guessing.

Furthermore, in the computation of the LDA classifier, we tried to leave the diagonal of the S_w matrix of covariances, in order to consider only the variances of the electrodes. This was done by setting all elements of the matrix to zero, except for the elements of the diagonal. However, no improvements were seen among all workflows. At this point, neither by normalizing using the full S_w matrix of covariances, nor by using the diagonal of it to only use the electrode variances brought important results in the search process of the theta ground truth. In the next paragraph, we reported the third possibility that we tried in this project, that is, not using the normalization at all.

Finally, two modifications brought improvements both for workflow 1 and workflow 2, not in terms of accuracies, but in terms of weight vectors interpretations. In fact, in both workflows, the weight vectors showed high scores corresponding to the frontal electrodes. Before feeding the preprocessed data without outliers to the LDA, for each participant, we averaged the data points corresponding to the same trial. In this way, data from the same trial became only one data point, losing information on the frequency class (freqc). This averaging procedures averages all data points in the theta range (4-8Hz, freqc=4,5,6,7). Doing so yields each data point still in the theta range, because all data points that were averaged were theta power values. After this, in the weight vector computation in LDA, the weight vector was set to $w = M_f \quad M_s$, avoiding the normalization we did before using the covariance matrix S_w. As a consequence, the classifier was only based on the mean electrodes difference between fail and success condition. According to the theta ground truth, in the success condition the theta power values are higher, the weight vector of LDA should show negative scores in the frontal area of the scalp. These scores should be negative because w is $M_f \quad M_s$ and not $M_s \quad M_f$. M_s has higher values for the frontal electrodes, and this should yield negative overall scores for w. Consequent improvements are reported in the results section.

2.6 A baseline classifier

Knowing that in the data there was the theta ground truth, we wanted to see if it was possible to achieve a higher accuracy than what we found using the Linear Discriminant Analysis classifier, when we applied the Leave One Out procedure. We programmed a baseline classifier with the idea that this should be better than the random classifier, because it was based on the theta ground truth. After the ANOVA Repeated Measures test, the theta ground truth was certainly contained in the data of the FCz electrode, meaning that in the successful trials the values of the FCz for any frequency range were higher than the same values, but for the fail trials. Based on this fact, we programmed the workflow of the baseline as shown below:

- for every participant ptp:
  - consider only the data of ptp
  - balance the fail and success classes
  - for each row of the participant dataframe:
    - leave row out
    - in the remaining rows, find the average value of FCz in the fail and success classes (avg_f, avg_s)
    - compute a cutoff value that is the average of avg_f and avg_s
    - classify the left out row: success if the FCz value is higher than the cutoff, otherwise as fail

– compute accuracies

3 Implementation

In this section, we include the description of the code we implemented, documenting how it was developed. We wrote all code in the python language, starting from what was developed in Galama, 2021. Considering that this research might be continued by a person in the future, we also styled the code to improve readability for a person who takes on this project. In fact, in the first phase of the research, we simplified the previous code and added annotation to all functions to increase comprehension. Functions were not type annotated, so we explicitly defined which type input and output parameters were. Because functions were also heavily dependent on a high number of parameters passed as arguments, we decreased the number of arguments for most functions. Indeed, these arguments were in the majority of cases redundant, and they could easily be obtained at the beginning of the function code.

Side effects were also present in the code, meaning that calls of a function modified global variables that are by definition defined outside the function itself, or in other cases parameters were passed to a function via assignment of a global variable. We removed these artifacts from the code, leading to a more functional programming paradigm.

To further improve readability, we split the python script that was originally in one big file into more files:

- **main.py** - it is the entry point of the python code. Here, the procedure functions such as running the LOO or PCA, contained in this file, are called whether their associated parameter in parameters.py tells doing so. For example, the function to run one round of LOO of LDA procedure, proc_run_loo_lda, is called if the boolean variable run_loo_lda is set to True in *parameters.py*.

- **parameters.py** modifiable parameters are in this file. For example, the amount of electrodes to use in the classification (used_electrodes) and parameters that make procedures running (e.g. parameters whose name starts with "run_").

- in **lda_procedures.py** we left the functions from Galama, 2021 that run the LDA on one participant's data (find_weights_LOO), on all participants (find_weights_LOO_all) and on all participants k times, averaging the results (find_weights_LOO_k_times). In this latter function, the LOO procedure is repeated k times in order to average the accuracies. In every iteration, the balancing process between fail and success data points will sample random data points from the majority set, success or fail data points, in order to match the cardinality of the minority set, fail or success data points. The k-times repetition has the purpose to average the different results given by this random sampling at every balancing step. In the workflows described in this report, this procedure has not been used, leaving $k = 1$, because we concentrated on achieving an accuracy higher than 50% in the first place.

- **new_lda.py** is the file where we programmed the new Linear Discriminant Analysis classifier class. In this way, we created a template of the classifier object that can be reused to implement other sorts of classifiers without adjusting other parts of the code. The class defines an LDA object with three functions: train, predict, test. The train function was called when we wanted to compute the LDA classifier. The predict function predicts the class fail or success, given a single data point (row of a dataframe data). The test function predicts all rows of a test dataframe and computes the accuracy of the LDA classifier. We said this template is reusable in the code because potentially we can program another class with the same three functions train, predict, test which implement a different classifier.

For instance, in this file, there is also the class LDA_sklearn that implements the default LDA classifier from the library scikit-learn.

- in the file **data_manage.py** we included functions that import data and functions to manipulate data. For the functions that import data, we also implemented a cache system that stores the imported data in temporary memory. In this way the data of all participants will be imported only once (in around 10 sec) and all subsequent calls to import functions take constant time. Regarding functions that manipulate data instead, we extracted duplicate code that was frequently used, in the following functions
  - *get_features_list* - given a dataframe it returns the list of all usable features for a classifier. In other words, it excludes column names which should not be used as features.
  - *get_participants_list* - given a dataframe, it returns the list of all participants whose data appear in the dataframe.
  - *get_averaged_data* - given a dataframe, it groups the data by participant, class and frequency range freqc and then averages the data in all groups. This was used to prepare the data for the ANOVA test.
  - *get_data_without_outliers* - it removes all rows with at least an outlier from a given dataframe. The default threshold for an outlier is set to five.
  - *get_data_without_outliers_cells* - it removes all cells values which are outliers from a given dataframe. The default threshold for an outlier is set to five.
  - *get_elec_x_freqc_search_space* - it pivots the dataframe on the freqc column in order to create a search space where the features are electrodes x frequency combinations. This inflates the number of columns of the dataframe while reducing the number of rows because the function brings information previously contained in the freqc column cells into the columns names which become in the form electrode_freqc (e.g. FCz_4, that means the values of the frequency power in the range 4-5Hz of the FCz electrode)
- All the workflows that have been used can be find in the python notebooks **N_lda*i*.ipynb**. For example, "N_lda1.ipynb" contains the workflow 1 with the Not Leave One Out procedure (NLOO), described in the methods section. All these notebooks are based on the functions defined in the above-mentioned python files.
- Besides the notebooks with the LDA procedures, we also included some notebooks to explain more the source code behind the plotting of the distribution of FCz (**N_distribution FCz.ipynb**), the outliers' removal (**N_outliers removal.ipynb**), and the averaging of the data in order to prepare them for a RM ANOVA test (**N_data_average.ipynb**).
- The **pca.py** file was left in the project folder. In this file, we originally programmed the Principal Component Analysis (PCA) algorithm to include it in the pre-processing phase of data.
- Also, the **optimal_subset.py** file was left in the project folder. In here, we left the original code that analyses the weight vectors found with LDA to find an optimal subset of electrodes for classification.

4 Results

4.1 Theta ground truth in the data

The first result we got is related to the presence of the theta ground truth in the data. Starting analyzing the frequency values of the data, every electrode value followed a positive skewed

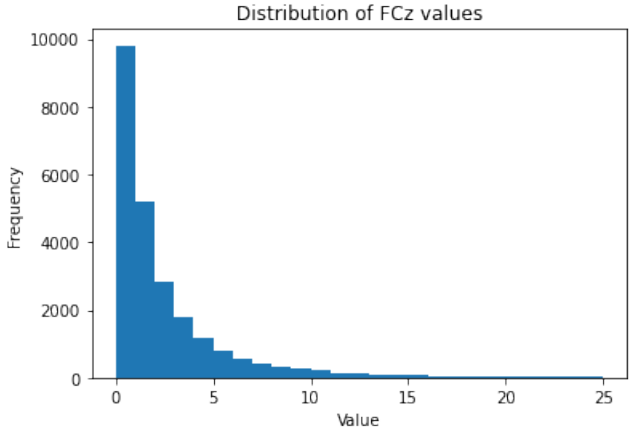


Figure 2: Distribution of FCz values from 0 to 25. All frequencies after the value 25 are cut out, because there is low to no presence of the value at all in the dataset. The maximum value of the FCz electrode is however 132418.063.

	ptp	class	freqc	Fp1	AF7	AF3	F1	...	P4	P6	P8	P10	PO8	PO4	O2
0	001	f	4	0.866	1.029	1.087	0.930	...	0.329	0.440	0.523	1.432	0.573	0.148	0.769
1	001	f	5	1.465	1.490	0.871	1.028	...	0.413	0.652	0.791	1.299	0.910	0.212	0.811
2	001	f	6	1.157	1.099	0.992	0.977	...	0.245	0.313	0.418	0.906	0.449	0.176	0.417
3	001	f	7	1.324	1.254	1.351	1.484	...	0.219	0.306	0.623	1.720	0.865	0.195	0.719
4	001	s	4	1.307	1.237	1.007	0.793	...	0.193	0.507	0.888	1.762	0.936	0.166	0.830
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
219	112	f	7	1.380	1.391	1.398	1.358	...	0.749	1.101	0.981	1.408	0.891	0.673	1.005
220	112	s	4	1.176	0.978	0.845	1.303	...	1.531	0.721	0.983	1.102	1.864	1.356	3.609
221	112	s	5	1.174	0.936	0.889	1.244	...	0.667	0.872	0.573	1.079	0.890	0.784	1.196
222	112	s	6	1.356	0.896	1.144	1.485	...	0.743	0.815	0.746	0.856	0.718	0.816	1.080
223	112	s	7	1.179	0.888	1.187	1.298	...	0.552	0.960	0.733	1.248	0.937	0.601	0.994

224 rows × 67 columns

Figure 3: Averaged data after the removal of the outliers

distribution. An example of this can be seen in figure 2. This analysis showed that for every electrode, there are data points with big numbers values. These are the outliers of the dataset, since it is not possible to measure such values. Considering them would increase the average frequency values, introducing noise in the analysis and in the training sets of the classifier. These numbers are weird artifacts in the imported data that were not taken into consideration in the past work.

Therefore, as specified in the methods, data points were removed where there was at least one electrode with value greater than five. During the removal of outliers, 12080 rows in the dataset were removed, leaving 13144 rows in the dataframe. Moreover, the criterion of outlier threshold set at five completely removed participant 018 from the dataset.

After we removed the outliers from the main data frame, we grouped the data by participant, frequency range (freqc), and class and averaged the rows within every group. The resulting dataframe was the one in figure 3, containing 224 rows: 28 rows for each participant, 4 rows for each frequency range (4-8Hz for the theta range), 2 rows per class (`f` or `s`). Therefore 28 4 2 = 224 rows. Finally, we pivoted the data on the freqc and class columns in order to bring the information relative to the frequency range and class in the dataframe columns, resulting in the dataframe of figure 4, ready for the statistical analysis.

At this point, we fed the data to a repeated measures ANOVA test. For the failsucc factor, the test yielded a significant difference in averages for success vs fail trials ($p < 0.001$) for each of the frequency ranges. The average difference was 0.098 between the success and fail condition. Results of the test can be seen in figure 5. Also, as it can be seen from the descriptive plot in

	ptp	AF3_4_f	AF3_4_s	AF3_5_f	AF3_5_s	AF3_6_f	AF3_6_s	...	TP8_4_s	TP8_5_f	TP8_5_s	TP8_6_f	TP8_6_s	TP8_7_f	TP8_7_s
0	001	1.087	1.007	0.871	0.989	0.992	1.102	...	1.094	0.868	1.259	0.617	0.974	0.654	1.023
1	002	1.098	0.405	0.928	1.232	1.119	1.157	...	1.525	0.773	1.466	1.128	0.864	1.009	0.678
2	003	1.192	0.838	1.126	0.989	1.016	1.219	...	0.769	0.877	0.874	0.938	0.652	0.461	0.720
3	004	1.199	1.369	1.242	1.028	1.080	1.198	...	0.928	0.698	0.584	0.551	0.618	0.511	0.528
4	005	1.051	1.082	1.013	1.149	0.964	0.850	...	0.884	0.730	0.733	0.832	0.643	0.622	0.706
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
23	108	1.001	0.947	1.140	0.932	0.976	1.086	...	0.360	0.314	0.311	0.300	0.274	0.249	0.253
24	109	0.847	0.969	0.805	0.927	0.886	0.862	...	0.315	0.246	0.271	0.247	0.222	0.209	0.186
25	110	1.005	0.943	0.849	0.914	1.160	1.130	...	0.361	0.365	0.422	0.306	0.348	0.322	0.288
26	111	0.721	0.920	0.801	0.987	0.856	0.985	...	0.674	0.344	0.355	0.323	0.348	0.300	0.291
27	112	1.327	0.845	0.952	0.889	1.251	1.144	...	0.590	2.103	0.916	0.820	0.479	0.945	0.750

28 rows × 513 columns

Figure 4: Pivoted dataframe on freqc and class after the averaging of data per participant, frequency range and class. The previous information that was contained in the columns freqc and class is now contained in the column names

Within Subjects Effects

Cases	Sum of Squares	df	Mean Square	F	p
failsucc	0.542	1	0.542	19.699	< .001
Residuals	0.743	27	0.028		
freqc	0.943[a]	3[a]	0.314[a]	4.353[a]	0.007[a]
Residuals	5.848	81	0.072		
failsucc ∗ freqc	0.019[a]	3[a]	0.006[a]	0.104[a]	0.957[a]
Residuals	4.853	81	0.060		

Figure 5: Within subjects effects of the repeated measures ANOVA. For the failsucc case, there is a significant difference for success vs fail trials ($p < 0.001$).

figure 6, the average values of the FCz electrode are higher in the success condition with respect to the fail condition, for all frequency ranges going from 4-5Hz to 7-8Hz.

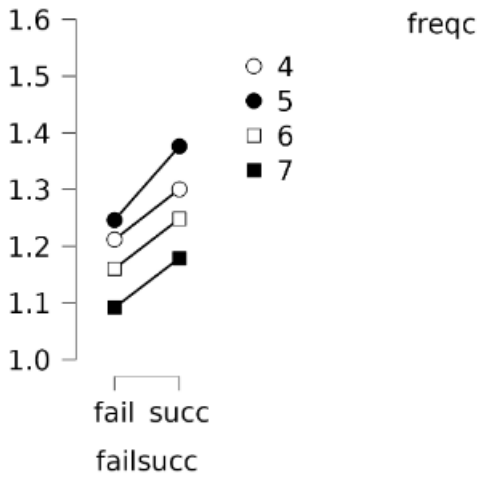


Figure 6: Descriptive plot of theta averages in the fail vs success condition, for each frequency range (freqc). In all cases, the theta power is higher in the successful trials

4.2 Performance of different workflows

As a result of the run of the workflows, we report here the results we obtained. Only in workflow 1 we obtained an accuracy of 74%. This was in fact the only case in which the test set was contained in the training set (because it was the same). In all other cases, within each participant, the LDA classifier was not able to generalize and classify a new data point. A resulting accuracy of around 50% was obtained from workflows 2 - 6. In table 2, we report the accuracies achieved for each participant in all the workflows we followed.

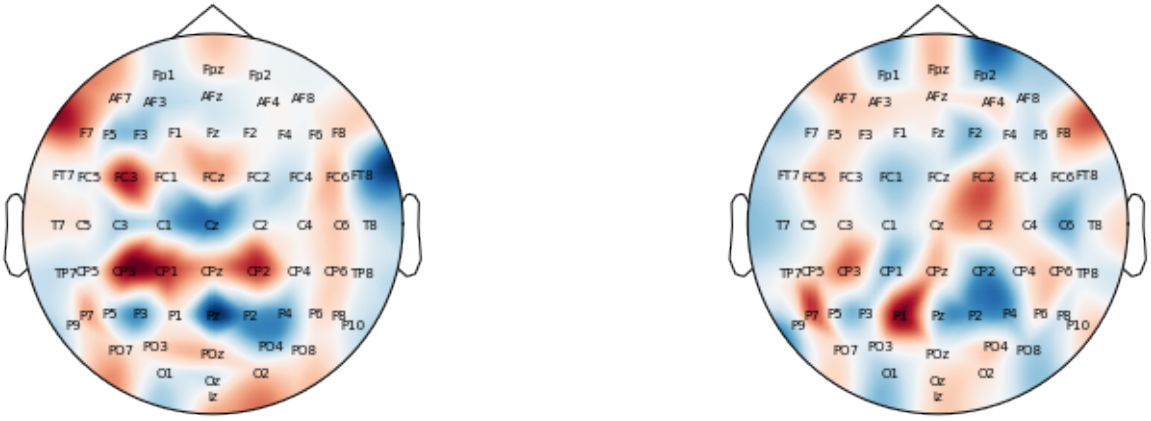
To understand why there was such a great drop in accuracy between workflow 1 and workflows 2-6, we analyzed the weight vectors computed in the different workflows by plotting their components on the scalp electrodes. A great difference came out in the visualization. In figure 7, there are the average weight vectors from workflow 1 and workflow 2 plotted on the scalp. The weight vectors elements were associated to the corresponding electrode. Different shades of red were used for positive number values, while different shades of blue were used for negative number values. The more intense red/blue, the more positive/negative was the value of the component.

This representation of figure 7, gave some insights in the results of the LDA. First, the average weight vectors differ a lot even though in workflow 2 at each step only one dataframe row is left out before the computation of the weight vector. One row out of hundreds should not influence so much the computation of the vector w, but from the visualization this is not the case. Also, from the two plots in figure 7, the LDA classifier focuses more on the parietal area (more intense shades of red and blue), rather than the frontal electrodes' area, that present more faded shades of colors. The more intense colors mean that those electrodes have greater importance on the classification because the weight that is given to them is further from zero. More positive means the electrode higher value contributes to a classification as "fail" and, vice versa, more negative means that the electrode higher value leads to a classification as "success". From section 2.3, it is not the other way around because a dataframe row is classified as "fail" if $w\ d > c$, where w is the weight vector and d the dataframe row. Therefore, positive weight vector components lead to a classification as "fail". Vice versa, negative weight vector components lead to a classification as "success" because this happens when $w\ \ d < c$.

Looking at the weight vectors per participant, in the majority of cases the visualization of the vectors computed by the LDA is very random, meaning that the classifier is not focusing on any electrode or small set of electrodes in particular. The shades of colors of the weight vector visualization seem randomly distributed across the scalp. The result of this is a weight

Participant	WF1	WF 2	WF 3	WF 4	WF 5	WF 6
001	75.0	49.07	52.53	56.82	58.43	54.17
002	100.0	56.67	48.38	36.84	45.37	48.33
003	82.14	47.32	50.83	51.32	46.11	48.21
004	66.4	46.64	55.92	55.41	59.21	47.63
005	64.95	50.98	51.79	55.41	50.0	48.77
009	70.26	55.88	54.48	47.22	50.0	46.73
010	59.01	45.04	49.86	48.9	48.91	50.55
011	81.61	52.3	55.86	54.69	50.0	52.87
012	65.18	49.46	51.31	43.02	51.16	54.11
013	100.0	46.88	53.97	63.64	51.76	0.0
014	100.0	42.59	54.56	43.18	50.52	50.0
015	73.04	46.08	52.41	38.6	46.02	51.96
016	63.46	52.69	51.6	46.74	54.26	54.11
017	73.14	53.72	51.18	46.61	57.43	45.45
019	70.32	52.67	49.52	44.12	33.33	43.05
020	77.72	51.63	54.65	48.11	48.72	55.98
021	75.93	54.81	49.59	53.91	44.23	52.22
022	68.25	49.74	51.08	47.5	55.98	51.06
101	63.36	52.05	50.66	52.78	45.06	50.68
103	65.98	56.02	53.24	48.39	50.0	52.7
104	93.75	50.0	52.08	50.0	44.64	43.75
106	67.71	51.43	55.12	50.0	50.0	50.86
107	61.83	52.7	51.16	48.91	44.76	52.59
108	60.29	46.35	49.31	54.03	47.86	46.88
109	60.66	52.79	51.75	42.06	50.0	47.46
110	64.44	48.71	52.02	55.11	49.07	46.34
111	62.4	51.02	52.56	47.6	52.69	52.56
112	95.0	47.5	53.07	41.43	48.03	47.5
Average:	73.64	50.46	50.38	49.01	49.55	48.09

Table 2: Table of accuracies achieved by the Linear Discriminant Analysis classifier for each participant across workflows.



(a) average weight vector from work ow 1

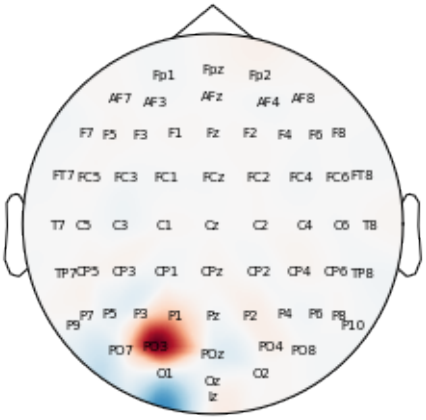
(b) average weight vector from work ow 2

Figure 7: Representation of the average weight vectors from workflow 1 (a), and from workflow 2 (b). The components of the vector were plotted on the scalp to the associated electrode. Different shades of red were used for positive number values, while different shades of blue were used for negative number values. The more intense red/blue, the more positive/negative was the value of the component.

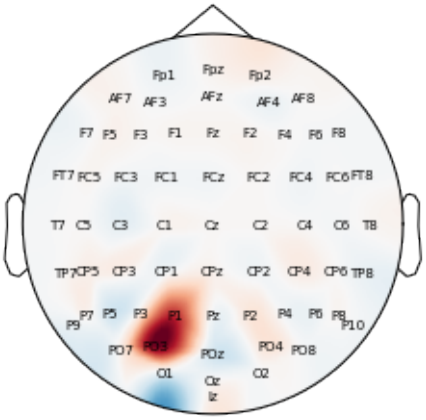
vector that looks like the average weight vector, from figure 7a or 7b. In other few cases, for example participant 001, the weight vector has a few components whose absolute value is very high. In figure 8, there are the weight vectors from workflows 1 and 2, of participant 001 only. In these, the classifier intensely focuses on the value of electrode PO3 for the classification. Because the weight component of PO3 is dark red, the higher the value of PO3, the higher will be the projection of w on the data point. As a consequence, a higher value of PO3 leads to a classification as "fail". Nevertheless, the average difference in values of PO3 between the two conditions (success and fail) is not statistically significant (p=0.646), so it is interesting to observe that the classifier strongly focuses on this variable to discriminate success from fail. The same effect is also seen for other participants. The classifier focused on a specific electrode in participants 009 (electrode Fz), participant 011 (electrode POz) and in participants 010,020,022 still on electrode PO3.

Another interesting observation, is that in the cases where the weight vector focuses on a small subset of electrodes, like in participant 001, the weight vector computed in workflow 1, slightly differs from the average weights vectors calculated in workflow 2, where the Leave One Out procedure is applied. These two weight vectors are almost the same, however in the second workflow, the average accuracy drops from 74% to 50%.

Even in workflow 6, where we imported only data of FCz, that we know contain the theta ground truth, the classifier could not score higher than around 50%. This workflow is interesting to analyze because the search space only contains one electrode, namely FCz. We decided to analyze the difference in average values of FCz between the success and fail condition, comparing it to the accuracy scored by the classifier, because a small difference between the conditions could be a reason why the classifier does not work well. The result is showed in figure 9. The plot presents the accuracies achieved by the classifier in workflow 6 versus the difference between the mean value in the success and fail condition. Overall, the average difference was minimal, with an average of 0.077 (std=0.143). Note that this difference is not the same as the one reported in the ANOVA results section (0.098). This is due to the balancing process. Running a two-sided t-test on the sample used to calculate the average difference of 0.077 yields a t-score of -0.77 with a p-value of 0.44. Therefore, with a significance level set to 5%, one could not reject the null hypothesis that this difference, however different from the ANOVA results, belongs to a



(a) average weight vector from work ow 1, ptp=001



(b) average weight vector from work ow 2, ptp=001

Figure 8: Representation of the weight vector for participant 001 in workflow 1 (a), and the average weight vector from workflow 2 (b). The components of the vector were plotted on the scalp to the associated electrode. Different shades of red were used for positive number values, while different shades of blue were used for negative number values. The more intense red/blue, the more positive/negative was the value of the component. When the Leave One Out procedure is applied (workflow 2 - b) the average weight vector slightly differs from the weight vector computed on the totality of data from participant 001 (a), and the average accuracy is 50%, not 74%.

distribution with the same mean. The maximum accuracy (62%) was achieved in participant 002 when there was the high difference of 0.371. The second-highest accuracy (55.4%) was achieved when there was the second-greatest difference of 0.319). There are also cases where the difference was negative, meaning that the average value of FCz was greater in the fail trials. In theory, according to the theta ground truth, the average values in the success conditions are greater than the ones in the fail conditions, however in the dataset this could not be the case because of the great standard deviation of the electrode values (std of FCz values=1.21).

Overall, in the interesting cases to consider, that are workflows 2 - 6, the accuracies are however not significant because are all behind the threshold of 70.0% from the table of significant accuracies. In any way, this was noticeable because they were all revolving around 50% which is the accuracy of the random classifier.

After that, we also tried the same workflows structures from 2 to 5, but on data of frontal electrodes only, and the following are the accuracies we obtained, again all non-significant: workflow 2 50%, workflow 3 51%, workflow 4 50%, workflow 5 49%.

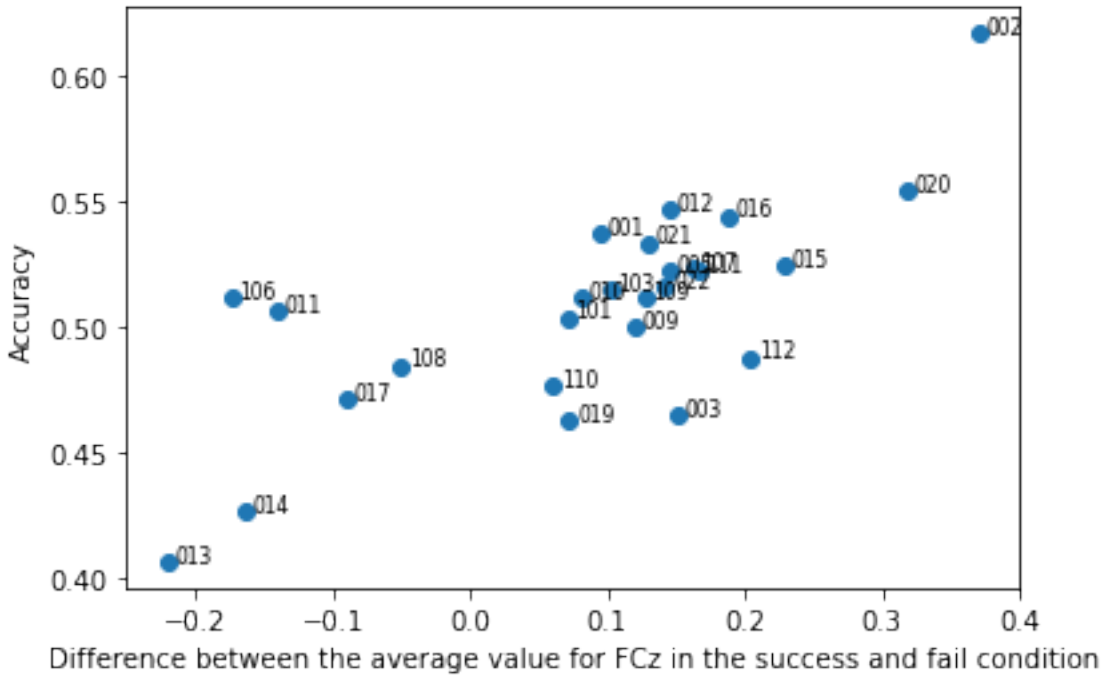
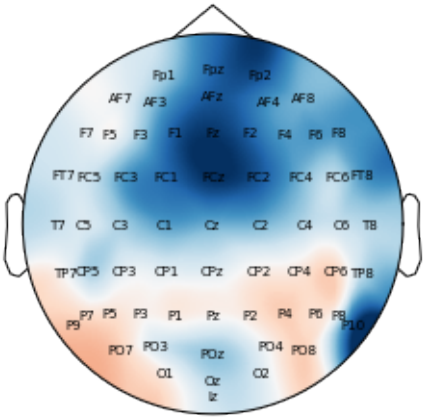


Figure 9: Accuracy vs. difference between average values of FCz across theta range in the success and fail condition. The data refer to the results from workflow 6.

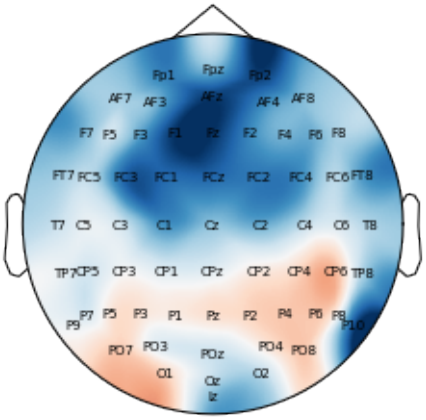
4.3 Modifications to the LDA and workflows to uncover the theta ground truth

When the data points coming from the same trial were averaged into a single data point, and the weight vector in the LDA computation was simply set to be $M_f \quad M_s$, without normalization, we got improvements both for workflow 1 (WF1) and workflow 2 (WF2). The respective accuracies were not better, but without the normalization, the theta ground truth emerges directly from the weight vector. This is because it is the difference of the mean electrode values between the two conditions, and in the success condition the values are higher in the frontal area of the scalp. A resulting accuracy around 60% was produced using WF1, while WF2 remained around 50%. The weight vector components are plotted on the scalp, figure 10, where it is evident that the frontal electrodes are of more importance for the prediction because they are drawn with a darker shade of blue. In WF1, the most important electrode is FCz, with the most negative component, that is the electrode with the most significant difference between conditions when we run the ANOVA test. Moreover, the weight vectors are very similar between WF1 and WF2, by looking at their average in figure 10a and 10b. Still, in WF2 the weight vector successfully identifies the most important variables to look at (frontal electrodes), but cannot score a higher accuracy than 50%. It seems that with the LOO, one left out data point is enough to produce a weight vector that is slightly different and cannot be used to predict an unseen data point.

It is important to note that is the average of weight vectors that shows the theta ground truth. When we consider weight vectors singularly per participant, the visualization focuses on many areas of the scalp, and there is no clear focus on the frontal area only. Sometimes, even the frontal area has positive scores (red) instead of the expected negative (blue).



(a) average weight vector from work ow 1, $w = M_f - M_s$



(b) average weight vector from work ow 2, $w = M_f - M_s$

Figure 10: Representation of the average weight vector calculated as $w = M_f \quad M_s$ in workflow 1 (a), and in workflow 2 (b). The components of the vector were plotted on the scalp to the associated electrode. Different shades of red were used for positive number values, while different shades of blue were used for negative number values. The more intense red/blue, the more positive/negative was the value of the component.

4.4 Performance of the baseline classifier

After the computation of the baseline classifier, described in section 2.6, this achieved an average accuracy of 53% across subjects, with a maximum accuracy of 57% for participant 002, and a minimum of 48% for participant 017. This showed that considering only the fact that the theta ground truth is contained in the values of FCz (significant difference, $p < .001$), a classification yields an average accuracy better than chance, if we base only on the difference from the average of the FCz values.

53% is not an accuracy far higher from the random classifier (50%), and this showed that only analyzing the difference between the average value in the success condition, versus the average value in the fail condition did not lead to a great result either, because this difference is minimal. In fact, the average difference between the FCz value in the success condition vs the fail condition is 0.09 (std: 0.10). Note that this difference is not equal to the one reported in section 4.2 (0.063, std=0.145). Every time we imported the data, we also applied the balancing process between the success and fail class, and random sampling from the majority class was done, so during different iterations the sampling extracted different data points which led to unequal mean differences. We also ran a two-sided t-test on this sample because the mean difference of 0.09 is not the same as the mean difference reported in the ANOVA results either (0.098). This computed a t-score of -0.42 with a p-value of 0.68. Also here, with a significance level of 5%, one could not reject the null hypothesis that this difference belongs to a distribution with the same mean.

Although the difference between the success and fail condition in the ANOVA test was significant, this difference is too small for a baseline classifier to work well because the standard deviation of FCz values is 1.04 and 1.08 respectively for the fail and success condition. With such great standard deviation in the values of the FCz electrode, this classifier cannot possibly classify well fail from success, considering only the theta ground truth of FCz, when the values it is considering differ so little (average=0.09, std=0.10) between the two conditions. Accordingly, when this difference is bigger, then the classifier can achieve higher accuracies. We reported in figure 11 the plot of the accuracies of the baseline classifier vs the difference between the average

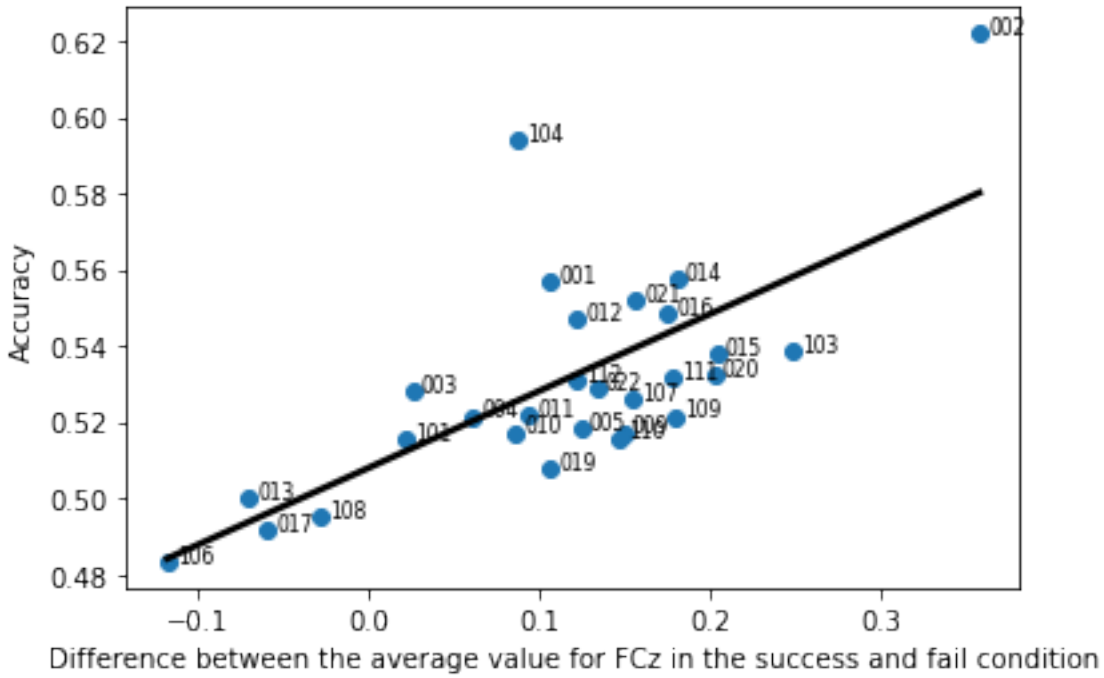


Figure 11: Accuracy of the baseline classifier vs. difference between average values of FCz in the success and fail condition. Every point is labeled with the participant number. In black, we represented the fitted regression line for the data, with a Pearson correlation coefficient of 0.79. Differences on the x-axis differ from the same differences calculated for the FCz values in figure 9. This is due to the random sampling during the balancing process, which randomly samples data points from the majority class.

values in the success and fail conditions. The plot presents a positive trend, with a Pearson correlation coefficient of 0.79.

5 Discussion

5.1 Theta ground truth in the data

As a result of the ANOVA test on the dataset without the outliers, there was a significant difference between the averages in the successful vs fail trials. Also, the higher average of theta power in the successful condition, as depicted in figure 6, replicates the theta ground truth as expected from Kandiah, 2020, because significant higher frontal theta power should appear in successful stops. This represented a good starting point, as the data contained the expected higher theta in successful inhibition. Not having it in the data would mean that some noise influenced the dataset too much, or in general that this fact would go against what found in Kandiah, 2020.

5.2 Workflows performance

The presence of the theta ground truth in the data gave the expectation of being able to improve the classifier. Nevertheless, as seen in the results of the workflows which have been implemented, the model was not able to generalize to incoming inputs never seen before (workflows 2-6). Only in the workflow 1 the LDA reached a higher accuracy, but this effect is probably happening because the classifier overfits on the training data, which are the same as the testing data in the first case. As a result, it can only classify well the training data. In all other workflows, including

the trials when we used frontal electrodes only, the average accuracy was always around 50%, meaning that the classifier did not reach a better state than the baseline random classifier.

The fact that there was a reduction in accuracy between workflow 1 and workflow 2-6 is quite unexpected. For instance, the only difference is that in workflow 2 only one row, out of hundreds in each participant dataframe, is left out before computing the weight vector. Such difference was not expected to yield the drop in accuracies in workflow 2. Indeed, there is no reason to believe that leaving only one row out, among hundreds (average=526 rows per participant), produces such a great difference in the computation of the weight vector, especially after the removal of the outliers. The weight vectors and the cutoff values should be nearly equal in the two workflows. In fact, even when the average weight vector in the two cases was very similar, figure 8, the results differ a lot (accuracy reduction to 50%).

From the results described in section 4, one possible explanation of this effect is that the average difference of electrode values in the two conditions, fail and success, is very small. At the same time, the standard deviation of electrode values is a lot larger than this average difference, and therefore the classifier could hardly tell from which distribution the electrode values came from. However, this explanation would not suffice to also explain why the accuracy is significantly higher than chance in workflow 1. The used data are the same also for workflow 1, but in this first scenario, the classifier can tell the difference between the fail and the success condition. The fact that could explain why the classifier can achieve high accuracies in workflow 1 is that the classifier had already seen every data point during the computation of LDA. During the test of workflow 1, the classifier is predicting the class of a point which was already separated during the computation phase, therefore there is no novelty in the test phase. In all other workflows, the classifier encounters novelty, as the data point that it tries to classify was never seen before, because it was left out. The minimal difference between the two conditions, with the relatively large standard deviation in the data, make a good classification not possible.

This last explanation would also elucidate why after modification to the LDA computation of section 2.5, the weight vector showing the theta ground truth in workflow 2, figure 10b, is not sufficient to correctly classify the data points better than chance. Here, the weight vector is slightly different from the weight vector of workflow 1, figure 10a, and this difference drops the accuracy from 60% to 50%.

6 Conclusions and future developments

In this work, we tried to classify successful from fail motor inhibition, analyzing the EEG data of the one second before the stop task. The analysis of this brain process is important in the study of attention, and such classifier can reveal more knowledge about the motor inhibition mechanics. Besides this, building a classifier which is able to classify if the brain is going to fail before any stop task even takes place, is useful material for applications that assist people to maintain concentration and avoid distractions. For example, a driver who is notified that, if at that moment a child were to cross the street, his brain would fail to stop the ongoing action, pressing the accelerator, in order to press the brake pedal.

At the very beginning of this project, we found that the data contained many outliers, and the theta ground truth would be revealed if we removed these outliers. After that, we brought modifications to the past work of Galama, 2021 in order to find the theta ground truth in the results from the classifier. We applied different workflows to look for a change in accuracy and weight vectors. Workflow 1 was the only one with an accuracy better than chance (73%), but also the only one where the training set was equal to the test set. In all other workflows, the Leave One Out procedure was applied and the left out data points were never classified correctly with an accuracy higher than 50%.

Eventually, one modification to the workflow and LDA computation was found to reveal the theta ground truth in the weight vector of the LDA. Averaging the data points corresponding

to the same trial, and computing the weight vector w without normalization, by setting it to $w = M_f \quad M_s$, was a successful improvement. Both in workflow 1 and 2 the average weight vector revealed a focus on the frontal area of the scalp, saying that the classifier was concentrated on the theta ground truth which deals with the frontal electrodes. This answers our research question, even though the accuracy of workflow 2 was not higher than chance. In WF2, the classifier can correctly identify the frontal electrodes as the most important one. On the other hand, it seems that one left out data point is enough to make the weight vector slightly different, making the prediction of an unknown point not better than random guessing. The fact that the classifier does not give a good prediction could be due to the minimal difference of success vs fail condition, together with the great standard deviation of electrode values, that makes a prediction hard.

At this point, we would like to point out some possibilities and facts that we think are worth to be considered in a future research.

- The modification that made possible the revelation of the theta ground truth was to avoid the normalization of the weight vector w using the S_w matrix of covariances. In a future research, this point should be taken into consideration to show why the LDA normalization of the weight vector hides the theta ground truth. We did a small trial where the matrix S_w was left with its diagonal only, setting all other elements to 0 in order to keep only the variances and deleting the covariances. Though, this type of normalization did not influence the accuracies, nor the weight visualization of theta ground truth either. So no type of normalization of the weight vector using the variances led to an improvement.

- One should consider the formula to calculate S_w to be the average of the other two within class scatter matrices: $S_w = (S_f + S_s)/2$. We computed all workflows using the sum instead of the average, but first trials of using the latter formula do not show improvements either.

- An interesting point to consider, after the improvements shown in section 4.3, is that the theta ground truth is shown in the averaged weight vector of all participants. In the majority of cases, if we look at the weight vector of a single participant, this does not present a focus on frontal electrodes. Sometimes, in w, there are positive components associated to the frontal electrodes, instead of negative. Sometimes, the w associates high importance to a subset of the electrodes which are not the frontal ones.

- Future research should give importance to the discovery of why predicting left out data points reduces the accuracy to random guessing. From workflow 1 to workflow 2, in fact, we noticed a drop from 73% accuracy to 50%. Also, after we average data points within the same trial and set $w = M_f \quad M_s$ and follow WF1,WF2 we noticed the drop from 60% to 50%.

- In this project, we adjusted the trade-off between the left-out data and the data used to estimate the weight vector. However, we decided to leave out 10% of the data points, not 10% of the trials. This is because one trial corresponds to four data points in the data frame, one for each theta frequency range (4,5,6,7). In the future, one could leave out 10% of the trials, in order to see if this adjustment really brings an improvement. Probably, leaving 10% of the data points out instead of 10% of the trials, had a minimal effect, since the data points are then averaged across theta range frequencies. In this way, only 2.5% of the trials are left out, after the averaging across data points which belong to the same trial.

- When we look at single weight vectors of some participants, both in WF1 and WF2, for some participants w has many high value components, suggesting that many variables need to be considered for a good prediction. For other participants, like participant 001 in

figure 8, the computed LDA focuses on a very restricted subset of electrodes, in this case $fPO3g$.

- In a future development of this work, one could also consider if the positive skewness of the electrode values is a factor that influences the outcome of the LDA computation and prediction (e.g. figure 2). A weak transformation like applying the square root to all values is enough to normalize the data. However, Fisher Discriminant Analysis, which was used in this work, does not require the normality assumption for the variables. Therefore, a normalization is not expected to yield better results.
- Also, importing the entire set of electrodes, or just a subset of them, could lead to better results. Considering different sets of electrodes could be another possibility for future research. We tried once to import a subset that included the frontal electrodes only, without better results in workflow 1 to 6, but maybe other groups of electrodes could bring an improvement.
- Finally, one could test if there are other thresholds for outliers definition that allow to retain more data points while preserving the theta ground truth.
- We also decided to include in this paper the methods that did not improve the classifier to enable a future development of this research, and show what did not work in this one. Besides that, the code was made more readable, optimized with respect to time, and side effects were removed, so that it is possible to combine it easily to continue working on the software.

References

Combrisson, E., & Jerbi, K. (2015). Exceeding chance level by chance: The caveat of theoretical chance levels in brain signal classification and statistical assessment of decoding accuracy. *Journal of neuroscience methods*, *250*, 126–136.

Galama, T. (2021). Bottom-up prediction of cognitive control features in eeg signals.

Jolliffe, I. T., & Cadima, J. (2016). Principal component analysis: A review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *374* (2065), 20150202.

Kandiah, S. (2020). Theta- and alpha-power dynamics related to response inhibition.

Kenemans, J. L., Schutte, I., Van Bijnen, S., & Logemann, H. N. A. (2022). How salience enhances inhibitory control: An analysis of electro-cortical mechanisms. *bioRxiv*. https://doi.org/10.1101/2022.01.12.475991

O'Connell, R. G., Dockree, P. M., Robertson, I. H., Bellgrove, M. A., Foxe, J. J., & Kelly, S. P. (2009). Uncovering the neural signature of lapsing attention: Electrophysiological signals predict errors up to 20 s before they occur. *Journal of Neuroscience*, *29* (26), 8604–8611. https://doi.org/10.1523/JNEUROSCI.5967-08.2009

Pandas dataframe: Pivot() function. (2022). https://pandas.pydata.org/docs/reference/api/pandas.pivot_table.html

Appendix A: Workflows

In this appendix, we attach all the workflows that have been run with the LDA procedure. The following is the list of all python notebooks where we explain the workflow, train the LDA classifier and test its performance. We summarize the features of all workflows described in the methods section in the following table:

Workflow	LOO	Outliers removal	Search space	Accuracy
1	✗	rows containing outlier	electrode values	74
2	✓	rows containing outlier	electrode values	50
3	✓	cells containing outlier	electrode values	52
4	✓	rows containing outlier	electrode x frequency combinations	51
5	✓	cells containing outlier	electrode x frequency combinations	51
6	✓	rows containing outlier	only data of FCz electrode	46

LDA workflow 1 (full training set for each participant)

In this notebook, we train an LDA classifier that used the entirety of data within each subject. In other words, we train the LDA on all the data available for each participant, without leaving out any dataset row to be used as test set. The purpose of this process is to show that such strategy yields an LDA classifier which is really accurate, with an avergae accuracy of 73%.

Workflow **summary**:

- Import the dataframe of all data
- Remove all the rows with at least one outlier (value > 5)
- For each participant ptp:
  - consider only the data of ptp
  - balance the number of fail and success rows
  - *train* the LDA classifier
  - *classify* all data of ptp using the weight vector **w** and the criterion **c**
  - average accuracies of all predictions and go to the next participant

First let's import the data of all participants and remove the outliers:

In [1]:

```
import parameters as params
import pandas as pd
from data_manage import import_all, get_data_without_outliers

full_data = import_all(params.used_electrodes, suppress_pbar=True)
data_no_out = get_data_without_outliers(full_data)

pd.set_option("display.max_rows", 10)
pd.set_option("display.max_columns", 15)

data_no_out
```

Out[1]:

	Fp1	AF7	AF3	F1	F3	F5	F7	...	PO8	PO4	O2	class	freqc	trial	ptp
0	0.539	0.436	0.524	0.575	0.433	0.141	0.909	...	0.990	0.069	1.144	f	4	0.0	001
1	1.337	1.619	0.396	0.406	0.762	0.656	0.864	...	0.164	0.069	0.377	f	5	0.0	001
2	0.360	1.997	0.510	0.747	0.361	0.971	1.644	...	0.126	0.005	0.324	f	6	0.0	001
3	3.118	3.213	1.150	2.278	1.603	1.670	1.818	...	2.954	0.706	2.220	f	5	1.0	001
4	0.063	0.411	0.200	0.128	0.064	0.210	0.146	...	0.001	0.223	0.324	f	6	1.0	001
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
13139	0.402	0.610	1.135	1.707	1.292	0.938	0.127	...	0.223	0.168	0.245	s	7	158.0	112
13140	0.173	0.116	0.568	0.314	0.780	0.799	0.720	...	1.427	0.706	3.078	s	7	159.0	112
13141	2.955	1.565	1.261	0.215	1.255	1.983	2.712	...	1.551	0.085	0.840	s	7	160.0	112
13142	1.271	1.267	2.070	2.108	2.693	1.507	0.538	...	0.403	0.216	0.375	s	7	166.0	112
13143	1.049	0.100	2.351	2.880	1.407	0.561	0.269	...	1.835	0.082	1.435	s	6	179.0	112

13144 rows × 68 columns

Now that we have the data without outliers let's train the new LDA classifier. For each participant, we train the LDA classifier on the totality of their data.

In [3]:

```
import new_lda
import pandas as pd
from data_manage import get_participants_list, get_balanced_data

import warnings
warnings.filterwarnings('ignore')

df_results = pd.DataFrame() # to store the results of the LDA procedure
for ptp in get_participants_list(data_no_out):
    # select only data of the participant ptp
    data_ptp = data_no_out[data_no_out["ptp"] == ptp]

    # balance the number of fail vs successful trials datapoints
    data_ptp_blnc = get_balanced_data(data_ptp)

    # train the lda model
    lda_classifier = new_lda.LDA()
    lda_classifier.train(data_ptp_blnc)
    # test the lda model
    results = lda_classifier.test(data_ptp_blnc)
    results_df = pd.DataFrame.from_dict(results, orient='index').T

    # summarize the results for each participant
    # over the entire data of the single participant
    summary_ptp = {
        "ptp": ptp,
        "acc": results_df["acc"].mean(),
        "correct": results_df["correct"].sum(),
        "incorrect": results_df["incorrect"].sum(),
        "n": results_df["n"].sum(),
    }

    summary_ptp_df = pd.DataFrame.from_dict(summary_ptp, orient='index').T
    df_results = pd.concat([df_results, summary_ptp_df], axis=0)

df_results.set_index("ptp", inplace=True)

df_results
```

Out[3]:

ptp	acc	correct	incorrect	n
001	0.75	162	54	216
002	1.0	60	0	60
003	0.848214	95	17	112
004	0.612648	310	196	506
005	0.688725	281	127	408
...	...	...	...	...
108	0.597656	459	309	768
109	0.624365	492	296	788
110	0.655172	304	160	464
111	0.616368	482	300	782
112	0.9875	79	1	80

28 rows × 4 columns

In [4]:

```
# calculate average accuracy from results
avg_acc = df_results["acc"].mean()
print(f"Average accuracy: {round(avg_acc, 3)}")
```

```
Average accuracy: 0.736
```

As we can see from the ouput of the code, proceeding in this way yields an accuracy of around 73%

LDA workflow 2 (no outlier rows + LOO)

In this notebook, for each participant we apply the Leave One Out procedure (LOO). After having imported the data, we remove the outliers by deleting all rows with at least one number greater than five (this is what we mean with no outliers rows). Then, for each participant, and for each row datapoint in the participant dataframe, we leave the trial out, train the LDA classifier on the rest of the data and then test the performance of the LDA classifier predicting the class of the left out row. Such procedure, called Leave One Out (LOO), yields an average accuracy of 50%.

Workflow **summary**:

- Import the dataframe of all data
- Remove all the rows with at least one outlier
- For each participant ptp:
  - consider only the data of ptp
  - balance the number of fail and success rows
  - For each row in the dataframe of ptp:
    - leave row out
    - *train* the LDA classifier on the rest of the rows
    - *classify* the left out row of ptp using the weight vector **w** and the criterion **c**
  - average accuracies of all predictions and go to the next participant

First let's import the data of all participants and remove the outliers:

In [1]:

```
import parameters as params
import pandas as pd
from data_manage import import_all, get_data_without_outliers

full_data = import_all(params.used_electrodes, suppress_pbar=True)
data_no_out = get_data_without_outliers(full_data)

pd.set_option("display.max_rows", 10)
pd.set_option("display.max_columns", 15)

data_no_out
```

Out[1]:

	Fp1	AF7	AF3	F1	F3	F5	F7	...	PO8	PO4	O2	class	freqc	trial	ptp
0	0.539	0.436	0.524	0.575	0.433	0.141	0.909	...	0.990	0.069	1.144	f	4	0.0	001
1	1.337	1.619	0.396	0.406	0.762	0.656	0.864	...	0.164	0.069	0.377	f	5	0.0	001
2	0.360	1.997	0.510	0.747	0.361	0.971	1.644	...	0.126	0.005	0.324	f	6	0.0	001
3	3.118	3.213	1.150	2.278	1.603	1.670	1.818	...	2.954	0.706	2.220	f	5	1.0	001
4	0.063	0.411	0.200	0.128	0.064	0.210	0.146	...	0.001	0.223	0.324	f	6	1.0	001
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
13139	0.402	0.610	1.135	1.707	1.292	0.938	0.127	...	0.223	0.168	0.245	s	7	158.0	112
13140	0.173	0.116	0.568	0.314	0.780	0.799	0.720	...	1.427	0.706	3.078	s	7	159.0	112
13141	2.955	1.565	1.261	0.215	1.255	1.983	2.712	...	1.551	0.085	0.840	s	7	160.0	112
13142	1.271	1.267	2.070	2.108	2.693	1.507	0.538	...	0.403	0.216	0.375	s	7	166.0	112
13143	1.049	0.100	2.351	2.880	1.407	0.561	0.269	...	1.835	0.082	1.435	s	6	179.0	112

13144 rows × 68 columns

Now that we have the data without outliers let's train the new LDA classifier. For each participant, and for each row in the participant dataframe, we leave one row out and train the LDA classifier on rest of the data.

In [3]:

```
import numpy as np
import pandas as pd
import new_lda
from sklearn.model_selection import LeaveOneOut # to perform the LOO procedure
from data_manage import get_participants_list, get_balanced_data

import warnings
warnings.filterwarnings('ignore')

df_results = pd.DataFrame() # to store the results of the LDA procedure
for ptp in get_participants_list(data_no_out):
    # select only data of the participant ptp
    data_ptp = data_no_out[data_no_out["ptp"] == ptp]

    # balance the number of fail and success trials to then train the LDA
    data_ptp_blnc = get_balanced_data(data_ptp)

    # run the LOO procedure
    loo_splitter = LeaveOneOut()
    results_ptp = pd.DataFrame() # to store results  for each leave out
    for train_index, test_index in loo_splitter.split(data_ptp_blnc):
        train_set = data_ptp_blnc.iloc[train_index]
        test_set = data_ptp_blnc.iloc[test_index]

        # train the LDA
        lda_classifier = new_lda.LDA()
        lda_classifier.train(train_set, balance_check=False)
        # test the LDA
        results = lda_classifier.test(test_set)
        results_df = pd.DataFrame.from_dict(results, orient='index').T

        results_ptp = pd.concat([results_ptp, results_df], axis=0)

    # summarize the results for each participant after one loo procedure
    # over the entire data of the single participant
    summary_ptp = {
        "ptp": ptp,
        "acc": results_ptp["acc"].mean(),
        "correct": results_ptp["correct"].sum(),
        "incorrect": results_ptp["incorrect"].sum(),
        "n": results_ptp["n"].sum(),
    }

    summary_ptp_df = pd.DataFrame.from_dict(summary_ptp, orient='index').T
    df_results = pd.concat([df_results, summary_ptp_df], axis=0)

df_results.set_index("ptp", inplace=True)

df_results
```

Out[3]:

ptp	acc	correct	incorrect	n
001	0.472222	102	114	216
002	0.616667	37	23	60
003	0.491071	55	57	112
004	0.474308	240	266	506
005	0.539216	220	188	408
...	...	...	...	...
108	0.458333	352	416	768
109	0.521574	411	377	788
110	0.493534	229	235	464
111	0.506394	396	386	782
112	0.425	34	46	80

28 rows × 4 columns

In [4]:

```
# calculate average accuracy from results
avg_acc = df_results["acc"].mean()
print(f"Average accuracy: {round(avg_acc, 3)}")
```

```
Average accuracy: 0.501
```

As we can see from the ouput of the code, proceeding in this way yields an accuracy of around 50%

LDA workflow 3 (no outlier cells + LOO)

In this notebook, for each participant we apply the Leave One Out procedure (LOO). After having imported the data, we remove the outliers by deleting cells with at a number greater than five (this is what we mean with no outliers cells). Removing only the cell value instead of the entire row can help us to preserve more data, assuming the cell outlier did not have an influence on also other cells values of the same row. Then, for each participant, and for each row datapoint in the participant dataframe, we leave the trial out, train the LDA classifier on the rest of the data and then test the performance of the LDA classifier predicting the class of the left out row. Such procedure, called Leave One Out (LOO), yields an average accuracy of 50%.

Workflow **summary**:

- Import the dataframe of all data
- Remove all cells with a value greater than five
- For each participant ptp:
  - consider only the data of ptp
  - fill the missing cell values with the *mean* of the other column values
  - balance the number of fail and success rows
  - For each row in the dataframe of ptp:
    - leave row out
    - *train* the LDA classifier on the rest of the rows
    - *classify* the left out row of ptp using the weight vector **w** and the criterion **c**
  - average accuracies of all predictions and go to the next participant

First let's import the data of all participants and remove the outliers:

In [1]:

```
import parameters as params
import pandas as pd
from data_manage import import_all, get_data_without_outliers_cells

full_data = import_all(params.used_electrodes, suppress_pbar=True)
data_no_out = get_data_without_outliers_cells(full_data)

pd.set_option("display.max_rows", 10)
pd.set_option("display.max_columns", 15)

data_no_out
```

Out[1]:

	Fp1	AF7	AF3	F1	F3	F5	F7	...	PO8	PO4	O2	class	freqc	trial	ptp
4	0.539	0.436	0.524	0.575	0.433	0.141	0.909	...	0.990	0.069	1.144	f	4	0.0	001
5	1.337	1.619	0.396	0.406	0.762	0.656	0.864	...	0.164	0.069	0.377	f	5	0.0	001
6	0.360	1.997	0.510	0.747	0.361	0.971	1.644	...	0.126	0.005	0.324	f	6	0.0	001
7	3.318	0.387	3.576	3.225	2.748	0.678	1.048	...	0.013	0.099	0.173	f	7	0.0	001
36	2.997	NaN	3.688	1.068	1.434	2.835	2.394	...	0.752	0.018	0.979	f	4	1.0	001
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9191	NaN	NaN	NaN	NaN	NaN	NaN	4.777	...	3.248	1.010	4.641	s	7	181.0	112
9220	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	2.082	0.980	NaN	s	4	182.0	112
9221	1.574	3.086	0.982	2.964	2.181	1.196	1.958	...	1.940	1.111	3.781	s	5	182.0	112
9222	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	2.705	0.121	1.403	s	6	182.0	112
9223	3.738	1.130	NaN	NaN	2.605	1.473	0.722	...	0.810	1.040	0.469	s	7	182.0	112

25224 rows × 68 columns

Now that we have the data without outliers let's train the new LDA classifier. For each participant, and for each row in the participant dataframe, we leave one row out and train the LDA classifier on rest of the data.

In [4]:

```
import numpy as np
import pandas as pd
import new_lda
from sklearn.model_selection import LeaveOneOut # to perform the LOO procedure
from data_manage import get_participants_list, get_balanced_data

import warnings
warnings.filterwarnings('ignore')

df_results = pd.DataFrame() # to store the results of the LDA procedure
for ptp in get_participants_list(data_no_out):
    # select only data of the participant ptp
    data_ptp = data_no_out[data_no_out["ptp"] == ptp]

    # fill the missing values
    data_ptp = data_ptp.fillna(data_ptp.mean())

    # balance the number of fail and success trials to then train the LDA
    data_ptp_blnc = get_balanced_data(data_ptp)

    # run the LOO procedure
    loo_splitter = LeaveOneOut()
    results_ptp = pd.DataFrame() # to store results  for each leave out
    for train_index, test_index in loo_splitter.split(data_ptp_blnc):
        train_set = data_ptp_blnc.iloc[train_index]
        test_set = data_ptp_blnc.iloc[test_index]

        # train the LDA
        lda_classifier = new_lda.LDA()
        lda_classifier.train(train_set, balance_check=False)
        # test the LDA
        results = lda_classifier.test(test_set)
        results_df = pd.DataFrame.from_dict(results, orient='index').T

        results_ptp = pd.concat([results_ptp, results_df], axis=0)

    # summarize the results for each participant after one loo procedure
    # over the entire data of the single participant
    summary_ptp = {
        "ptp": ptp,
        "acc": results_ptp["acc"].mean(),
        "correct": results_ptp["correct"].sum(),
        "incorrect": results_ptp["incorrect"].sum(),
        "n": results_ptp["n"].sum(),
    }

    summary_ptp_df = pd.DataFrame.from_dict(summary_ptp, orient='index').T
    df_results = pd.concat([df_results, summary_ptp_df], axis=0)

df_results.set_index("ptp", inplace=True)

df_results
```

Out[4]:

ptp	acc	correct	incorrect	n
001	0.525281	374	338	712
002	0.493056	213	219	432
003	0.495833	357	363	720
004	0.550987	335	273	608
005	0.480519	296	320	616
...	...	...	...	...
108	0.499016	507	509	1016
109	0.509346	436	420	856
110	0.526882	392	352	744
111	0.523622	532	484	1016
112	0.495283	420	428	848

29 rows × 4 columns

In [5]:

```
# calculate average accuracy from results
avg_acc = df_results["acc"].mean()
print(f"Average accuracy: {round(avg_acc, 3)}")
```

```
Average accuracy: 0.517
```

As we can see from the ouput of the code, proceeding in this way yields an accuracy of around 50%

LDA workflow 4 (no outlier rows + electrode x freq ranges search space + LOO)

In this notebook, for each participant we apply the Leave One Out procedure (LOO). After having imported the data, we remove the outliers by deleting all rows with at least one number greater than five (this is what we mean with no outliers rows). In this case, we pivot the dataframe on the freqc column (frequency range) in order to get an electrode x frequency search space. Then, for each participant, and for each row datapoint in the participant dataframe, we leave the trial out, train the LDA classifier on the rest of the data and then test the performance of the LDA classifier predicting the class of the left out row. Such procedure, called Leave One Out (LOO), yields an average accuracy of 50%.

Workflow **summary**:

- Import the dataframe of all data
- Remove all the rows with at least one outlier
- For each participant ptp:
  - consider only the data of ptp
  - pivot the ptp dataframe on the freqc column (frequency range) in order to create an electrode x frequency search space
  - fill the missing cell values with the *mean* of the other column values
  - balance the number of fail and success rows
  - For each row in the dataframe of ptp:
    - leave row out
    - *train* the LDA classifier on the rest of the rows
    - *classify* the left out row of ptp using the weight vector **w** and the criterion **c**
  - average accuracies of all predictions and go to the next participant

First let's import the data of all participants and remove the outliers:

In [1]:

```
import parameters as params
import pandas as pd
from data_manage import import_all, get_data_without_outliers

full_data = import_all(params.used_electrodes, suppress_pbar=True)
data_no_out = get_data_without_outliers(full_data)

pd.set_option("display.max_rows", 10)
pd.set_option("display.max_columns", 15)

data_no_out
```

Out[1]:

	Fp1	AF7	AF3	F1	F3	F5	F7	...	PO8	PO4	O2	class	freqc	trial	ptp
0	0.539	0.436	0.524	0.575	0.433	0.141	0.909	...	0.990	0.069	1.144	f	4	0.0	001
1	1.337	1.619	0.396	0.406	0.762	0.656	0.864	...	0.164	0.069	0.377	f	5	0.0	001
2	0.360	1.997	0.510	0.747	0.361	0.971	1.644	...	0.126	0.005	0.324	f	6	0.0	001
3	3.118	3.213	1.150	2.278	1.603	1.670	1.818	...	2.954	0.706	2.220	f	5	1.0	001
4	0.063	0.411	0.200	0.128	0.064	0.210	0.146	...	0.001	0.223	0.324	f	6	1.0	001
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
13139	0.402	0.610	1.135	1.707	1.292	0.938	0.127	...	0.223	0.168	0.245	s	7	158.0	112
13140	0.173	0.116	0.568	0.314	0.780	0.799	0.720	...	1.427	0.706	3.078	s	7	159.0	112
13141	2.955	1.565	1.261	0.215	1.255	1.983	2.712	...	1.551	0.085	0.840	s	7	160.0	112
13142	1.271	1.267	2.070	2.108	2.693	1.507	0.538	...	0.403	0.216	0.375	s	7	166.0	112
13143	1.049	0.100	2.351	2.880	1.407	0.561	0.269	...	1.835	0.082	1.435	s	6	179.0	112

13144 rows × 68 columns

Now that we have the data without outliers let's train the new LDA classifier. For each participant, and for each row in the participant dataframe, we leave one row out and train the LDA classifier on rest of the data.

In [6]:

```
import pandas as pd
import new_lda
from sklearn.model_selection import LeaveOneOut # to perform the LOO procedure
from data_manage import get_participants_list, get_balanced_data, get_elec_x_freqc_search_space

import warnings
warnings.filterwarnings('ignore')

df_results = pd.DataFrame() # to store the results of the LDA procedure
for ptp in get_participants_list(data_no_out):
    # select only data of the participant ptp
    data_ptp = data_no_out[data_no_out["ptp"] == ptp]

    # pivot the df on freqc in order to have columns as electrode_freqc
    # this increases the number of columns while reducing the number of rows
    # because it's creating the electrode x frequency search space
    data_ptp_pivoted = get_elec_x_freqc_search_space(data_ptp)

    # fill the missing values
    data_ptp_pivoted = data_ptp_pivoted.fillna(data_ptp_pivoted.mean())

    # balance the number of fail and success trials to then train the LDA
    data_ptp_blnc = get_balanced_data(data_ptp_pivoted)

    # run the LOO procedure
    loo_splitter = LeaveOneOut()
    results_ptp = pd.DataFrame() # to store results  for each leave out
    for train_index, test_index in loo_splitter.split(data_ptp_blnc):
        train_set = data_ptp_blnc.iloc[train_index]
        test_set = data_ptp_blnc.iloc[test_index]

        # train the LDA: we used the sklearn implementation to avoid no convergence
        lda_classifier = new_lda.LDA_sklearn()
        lda_classifier.train(train_set, balance_check=False)
        # test the LDA
        results = lda_classifier.test(test_set)
        results_df = pd.DataFrame.from_dict(results, orient='index').T

        results_ptp = pd.concat([results_ptp, results_df], axis=0)

    # summarize the results for each participant after one loo procedure
    # over the entire data of the single participant
    summary_ptp = {
        "ptp": ptp,
        "acc": results_ptp["acc"].mean(),
        "correct": results_ptp["correct"].sum(),
        "incorrect": results_ptp["incorrect"].sum(),
        "n": results_ptp["n"].sum(),
    }

    summary_ptp_df = pd.DataFrame.from_dict(summary_ptp, orient='index').T
    df_results = pd.concat([df_results, summary_ptp_df], axis=0)

df_results.set_index("ptp", inplace=True)

df_results
```

Out[6]:

ptp	acc	correct	incorrect	n
001	0.583333	77.0	55.0	132.0
002	0.421053	16.0	22.0	38.0
003	0.486842	37.0	39.0	76.0
004	0.567568	84.0	64.0	148.0
005	0.472973	70.0	78.0	148.0
...	...	...	...	...
108	0.532258	132.0	116.0	248.0
109	0.46729	100.0	114.0	214.0
110	0.579545	102.0	74.0	176.0
111	0.496	124.0	126.0	250.0
112	0.442857	31.0	39.0	70.0

28 rows × 4 columns

In [3]:

```
# calculate average accuracy from results
avg_acc = df_results["acc"].mean()
print(f"Average accuracy: {round(avg_acc, 3)}")
```

```
Average accuracy: 0.515
```

As we can see from the ouput of the code, proceeding in this way yields an accuracy of around 50%

LDA workflow 5 (LOO + no outlier cells + electrode x freq ranges search space)

In this notebook, for each participant we apply the Leave One Out procedure (LOO). After having imported the data, we remove the outliers by deleting cells with a number greater than five (this is what we mean with no outliers cells). Removing only the cell value instead of the entire row can help us to preserve more data, assuming the cell outlier did not have an influence on also other cells values of the same row. In this case, we pivot the dataframe on the freqc column (frequency range) in order to get an electrode x frequency search space. Then, for each participant, and for each row datapoint in the participant dataframe, we leave the trial out, train the LDA classifier on the rest of the data and then test the performance of the LDA classifier predicting the class of the left out row. Such procedure, called Leave One Out (LOO), yields an average accuracy of 50%.

Workflow **summary**:

- Import the dataframe of all data
- Remove all cells with a value greater than five
- For each participant ptp:
  - consider only the data of ptp
  - pivot the ptp dataframe on the freqc column (frequency range) in order to create an electrode x frequency search space
  - fill the missing cell values with the *mean* of the other column values
  - balance the number of fail and success rows
  - For each row in the dataframe of ptp:
    - leave row out
    - *train* the LDA classifier on the rest of the rows
    - *classify* the left out row of ptp using the weight vector **w** and the criterion **c**
  - average accuracies of all predictions and go to the next participant

First let's import the data of all participants and remove the outliers:

In [3]:

```
import parameters as params
import pandas as pd
from data_manage import import_all, get_data_without_outliers_cells

full_data = import_all(params.used_electrodes, suppress_pbar=True)
data_no_out = get_data_without_outliers_cells(full_data)

pd.set_option("display.max_rows", 10)
pd.set_option("display.max_columns", 15)

data_no_out
```

Out[3]:

	Fp1	AF7	AF3	F1	F3	F5	F7	...	PO8	PO4	O2	class	freqc	trial	ptp
4	0.539	0.436	0.524	0.575	0.433	0.141	0.909	...	0.990	0.069	1.144	f	4	0.0	001
5	1.337	1.619	0.396	0.406	0.762	0.656	0.864	...	0.164	0.069	0.377	f	5	0.0	001
6	0.360	1.997	0.510	0.747	0.361	0.971	1.644	...	0.126	0.005	0.324	f	6	0.0	001
7	3.318	0.387	3.576	3.225	2.748	0.678	1.048	...	0.013	0.099	0.173	f	7	0.0	001
36	2.997	NaN	3.688	1.068	1.434	2.835	2.394	...	0.752	0.018	0.979	f	4	1.0	001
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9191	NaN	NaN	NaN	NaN	NaN	NaN	4.777	...	3.248	1.010	4.641	s	7	181.0	112
9220	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	2.082	0.980	NaN	s	4	182.0	112
9221	1.574	3.086	0.982	2.964	2.181	1.196	1.958	...	1.940	1.111	3.781	s	5	182.0	112
9222	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	2.705	0.121	1.403	s	6	182.0	112
9223	3.738	1.130	NaN	NaN	2.605	1.473	0.722	...	0.810	1.040	0.469	s	7	182.0	112

25224 rows × 68 columns

Now that we have the data without outliers let's train the new LDA classifier. For each participant, and for each row in the participant dataframe, we leave one row out and train the LDA classifier on rest of the data.

In [4]:

```
import pandas as pd
import new_lda
from sklearn.model_selection import LeaveOneOut # to perform the LOO procedure
from data_manage import get_participants_list, get_balanced_data, get_elec_x_freqc_search_space

import warnings
warnings.filterwarnings('ignore')

df_results = pd.DataFrame() # to store the results of the LDA procedure
for ptp in get_participants_list(data_no_out):
    # select only data of the participant ptp
    data_ptp = data_no_out[data_no_out["ptp"] == ptp]

    # pivot the df on freqc in order to have columns as electrode_freqc
    # this increases the number of columns while reducing the number of rows
    # because it's creating the electrode x frequency search space
    data_ptp_pivoted = get_elec_x_freqc_search_space(data_ptp)

    # fill the missing values
    data_ptp_pivoted = data_ptp_pivoted.fillna(data_ptp_pivoted.mean())

    # balance the number of fail and success trials to then train the LDA
    data_ptp_blnc = get_balanced_data(data_ptp_pivoted)

    # run the LOO procedure
    loo_splitter = LeaveOneOut()
    results_ptp = pd.DataFrame() # to store results  for each leave out
    for train_index, test_index in loo_splitter.split(data_ptp_blnc):
        train_set = data_ptp_blnc.iloc[train_index]
        test_set = data_ptp_blnc.iloc[test_index]

        lda_classifier = new_lda.LDA()
        lda_classifier.train(train_set, balance_check=False)
        results = lda_classifier.test(test_set)
        results_df = pd.DataFrame.from_dict(results, orient='index').T

        results_ptp = pd.concat([results_ptp, results_df], axis=0)

    # summarize the results for each participant after one loo procedure
    # over the entire data of the single participant
    summary_ptp = {
        "ptp": ptp,
        "acc": results_ptp["acc"].mean(),
        "correct": results_ptp["correct"].sum(),
        "incorrect": results_ptp["incorrect"].sum(),
        "n": results_ptp["n"].sum(),
    }

    summary_ptp_df = pd.DataFrame.from_dict(summary_ptp, orient='index').T
    df_results = pd.concat([df_results, summary_ptp_df], axis=0)

df_results.set_index("ptp", inplace=True)

df_results
```

Out[4]:

ptp	acc	correct	incorrect	n
001	0.561798	100	78	178
002	0.490741	53	55	108
003	0.444444	80	100	180
004	0.565789	86	66	152
005	0.611842	93	59	152
...	...	...	...	...
108	0.480315	122	132	254
109	0.462617	99	115	214
110	0.532258	99	87	186
111	0.503937	128	126	254
112	0.396226	84	128	212

29 rows × 4 columns

In [ ]:

```
# calculate average accuracy from results
avg_acc = df_results["acc"].mean()
print(f"Average accuracy: {round(avg_acc, 3)}")
```

```
Average accuracy: 0.515
```

As we can see from the ouput of the code, proceeding in this way yields an accuracy of around 50%

LDA workflow 6 (only FCz electrode + no outlier rows + LOO)

In this notebook, for each participant we apply the Leave One Out procedure (LOO). We import only data of the FCz electrode which we know presents the theta ground truth after running a statistical ANOVA test. We remove the outliers by deleting rows with a number greater than five (this is what we mean with no outliers rows). Then, for each participant, and for each row datapoint in the participant dataframe, we leave the trial out, train the LDA classifier on the rest of the data and then test the performance of the LDA classifier predicting the class of the left out row. Such procedure yields an average accuracy of 50%.

Workflow **summary**:

- Import the dataframe of FCz data of all participants
- Remove all rows with a value greater than five
- For each participant ptp:
  - consider only the data of ptp
  - balance the number of fail and success rows
  - For each row in the dataframe of ptp:
    - leave row out
    - *train* the LDA classifier on the rest of the rows
    - *classify* the left out row of ptp using the weight vector **w** and the criterion **c**
  - average accuracies of all predictions and go to the next participant

First let's import the data of all participants and remove the outliers:

In [1]:

```
import parameters as params
import pandas as pd
from data_manage import import_all, get_data_without_outliers

full_data = import_all(['FCz'], suppress_pbar=True)
data_no_out = get_data_without_outliers(full_data)

pd.set_option("display.max_rows", 10)
pd.set_option("display.max_columns", 15)

data_no_out
```

Out[1]:

	FCz	class	freqc	trial	ptp
0	0.674	f	4	0.0	001
1	0.179	f	5	0.0	001
2	0.322	f	6	0.0	001
3	3.092	f	7	0.0	001
4	0.158	f	4	1.0	001
...	...	...	...	...	...
20785	0.389	s	7	178.0	112
20786	3.751	s	4	179.0	112
20787	4.291	s	6	179.0	112
20788	3.200	s	4	181.0	112
20789	1.209	s	7	182.0	112

20790 rows × 5 columns

Now that we have the data without outliers let's train the new LDA classifier. For each participant, and for each row in the participant dataframe, we leave one row out and train the LDA classifier on rest of the data.

In [3]:

```
import pandas as pd
import new_lda
from sklearn.model_selection import LeaveOneOut # to perform the LOO procedure
from data_manage import get_participants_list, get_balanced_data

import warnings
warnings.filterwarnings('ignore')

df_results = pd.DataFrame() # to store the results of the LDA procedure
for ptp in get_participants_list(data_no_out):
    # select only data of the participant ptp
    data_ptp = data_no_out[data_no_out["ptp"] == ptp]

    # balance the number of fail and success trials to then train the LDA
    data_ptp_blnc = get_balanced_data(data_ptp)

    # run the LOO procedure
    loo_splitter = LeaveOneOut()
    results_ptp = pd.DataFrame() # to store results  for each leave out
    for train_index, test_index in loo_splitter.split(data_ptp_blnc):
        train_set = data_ptp_blnc.iloc[train_index]
        test_set = data_ptp_blnc.iloc[test_index]

        # train the LDA: we used the sklearn implementation to avoid no convergence
        lda_classifier = new_lda.LDA_sklearn()
        lda_classifier.train(train_set, balance_check=False)
        # test the LDA
        results = lda_classifier.test(test_set)
        results_df = pd.DataFrame.from_dict(results, orient='index').T

        results_ptp = pd.concat([results_ptp, results_df], axis=0)

    # summarize the results for each participant after one loo procedure
    # over the entire data of the single participant
    summary_ptp = {
        "ptp": ptp,
        "acc": results_ptp["acc"].mean(),
        "correct": results_ptp["correct"].sum(),
        "incorrect": results_ptp["incorrect"].sum(),
        "n": results_ptp["n"].sum(),
    }

    summary_ptp_df = pd.DataFrame.from_dict(summary_ptp, orient='index').T
    df_results = pd.concat([df_results, summary_ptp_df], axis=0)

df_results.set_index("ptp", inplace=True)

df_results
```

Out[3]:

ptp	acc	correct	incorrect	n
001	0.396721	242.0	368.0	610.0
002	0.457921	185.0	219.0	404.0
003	0.506667	228.0	222.0	450.0
004	0.498294	292.0	294.0	586.0
005	0.524648	298.0	270.0	568.0
...	...	...	...	...
108	0.365011	338.0	588.0	926.0
109	0.508454	421.0	407.0	828.0
110	0.458824	312.0	368.0	680.0
111	0.519588	504.0	466.0	970.0
112	0.475	171.0	189.0	360.0

29 rows × 4 columns

In [4]:

```
# calculate average accuracy from results
avg_acc = df_results["acc"].mean()
print(f"Average accuracy: {round(avg_acc, 3)}")
```

```
Average accuracy: 0.463
```

As we can see from the ouput of the code, proceeding in this way yields an accuracy of around 50%

Appendix B: LDA classifiers

In this second appendix, we report the code of the two LDA classes used as classifiers. The first class LDA is the class that implements the LDA algorithm described in section 2.3. The second class, LDA_2 is the LDA classifier with the modifications described in section 2.5.

```
class LDA:

    """
    This class implements the LDA algorithm.
    """

    def __init__(self) -> None:
        self.w = None # weight vector
        self.cutoff = None # cutoff value

    def train(self, dataset: pd.DataFrame, balance_check=True) -> None:

        """
        This method computes the LDA model, calculating the weight vector
        and the cutoff.
        """
        f_no = dataset[dataset['class'] == 'f'].shape[0]
        s_no = dataset[dataset['class'] == 's'].shape[0]
        if balance_check and f_no != s_no: print(f"WARN: f and s trials not
            ↪ balanced! {f_no} f, {s_no} s")

        # drop every column that is not a feature
        if "freqc" in dataset.columns: dataset = dataset.drop(columns=['freqc'
            ↪ ])
        if "trial" in dataset.columns: dataset = dataset.drop(columns=['trial'
            ↪ ])
        if "ptp" in dataset.columns: dataset = dataset.drop(columns=['ptp'])

        # calculate the mean vectors for fail and success classes
        mean_vector_fail = dataset[dataset['class'] == 'f'].mean(axis=0,
            ↪ numeric_only=True)
        mean_vector_success = dataset[dataset['class'] == 's'].mean(axis=0,
            ↪ numeric_only=True)

        # calculate the scatter matrices for fail and success classes
        scatter_matrix_fail = dataset[dataset['class'] == 'f'].cov()
        scatter_matrix_success = dataset[dataset['class'] == 's'].cov()

        S_w = scatter_matrix_fail + scatter_matrix_success

        # calculate the pseudo inverse of S_w
        S_w_inv = np.linalg.pinv(S_w)

        # calculate the weight vector
        self.w = np.dot(S_w_inv, mean_vector_fail - mean_vector_success)
```

```
        # calculate the cutoff value
        self.cutoff = self.w.T.dot((mean_vector_fail + mean_vector_success)) /
            ↪ 2

    def predict(self, data_row: pd.DataFrame) -> list:
        """
        This method predicts the class given one datapoint
        """

        if self.w is None: raise Exception("LDA model not yet trained!")
        if "class" in data_row: data_row.drop("class", inplace=True)
        if "freqc" in data_row: data_row.drop("freqc", inplace=True)
        if "trial" in data_row: data_row.drop("trial", inplace=True)
        if "ptp" in data_row: data_row.drop("ptp", inplace=True)

        # calculate the row projection
        data_row = np.array(data_row)
        projection = data_row.dot(self.w)

        # predict
        if projection < self.cutoff:
            return 's' # classify as success
        else:
            return 'f' # classify as fail

    def test(self, test_set: pd.DataFrame) -> dict:

        """
        Given a test set this method tests the performance of the LDA model
        """

        y_true, y_pred = list(), list()

        for _, row in test_set.iterrows():
            y_true.append(row['class']) # save the true label
            y_pred.append(self.predict(row)) # save the predicted label

        results = dict()
        results["acc"] = metrics.accuracy_score(y_true, y_pred)
        results["b_acc"] = metrics.balanced_accuracy_score(y_true, y_pred)
        results["correct"] = metrics.accuracy_score(y_true, y_pred, normalize=
            ↪ False)
        results["incorrect"] = len(y_true) - results["correct"]
        results["n"] = len(y_true)
        results["w"] = self.w
        results["c"] = self.cutoff

        return results
```

```
class LDA_2:
```

```
"""
This class implements the LDA algorithm.
It's the same as LDA but with a few changes:

1) elements of the same trials are averaged
2) the weight vector is simply the difference between Mf and Ms

UNUSED changes:
3) instead of computing covariance matrices we estimate them computing
    ‹! scatter matrices
4) the within class scatter matrix is the average between Sf and Ss, not
    ‹! the sum
5) the within class scatter matrix is then made diagonal to consider only
    ‹! the variance of the electrodes, not the covariances
    (all the elements outside the diagonal are set to 0)
"""

def __init__(self) -> None:
    self.w = None # weight vector
    self.cutoff = None # cutoff value

def train(self, dataset: pd.DataFrame, balance_check=True) -> None:

    """
    This method trains the LDA model, calculating the weight vector
    and the cutoff.
    """
    dataset_f = dataset[dataset['class'] == 'f'].copy()
    dataset_s = dataset[dataset['class'] == 's'].copy()

    f_no = dataset_f.shape[0]
    s_no = dataset_s.shape[0]
    if balance_check and f_no != s_no: print(f"WARN: f and s trials not
        ‹! balanced! {f_no} f, {s_no} s")

    # drop every column that is not a feature
    if "ptp" in dataset.columns: dataset = dataset.drop(columns=['ptp'])

    # average the data of the same trial number, then remove freqc because
        ‹! it's not needed anymore
    dataset_f = dataset_f.groupby('trial').mean()
    dataset_s = dataset_s.groupby('trial').mean()
    if "trial" in dataset_f.columns: dataset_f = dataset_f.drop(columns=['
        ‹! trial'])
    if "trial" in dataset_s.columns: dataset_s = dataset_s.drop(columns=['
        ‹! trial'])
    if "freqc" in dataset_f.columns: dataset_f = dataset_f.drop(columns=['
        ‹! freqc'])
    if "freqc" in dataset_s.columns: dataset_s = dataset_s.drop(columns=['
        ‹! freqc'])
```

```
        # calculate the mean vectors for fail and success classes
        mean_vector_fail = dataset_f.mean(axis=0, numeric_only=True)
        mean_vector_success = dataset_s.mean(axis=0, numeric_only=True)

        # UNCOMMENT CODE TO NORMALIZE WEIGHT VECTOR WITH ELECTRODE VARIANCES
        # # calculate the scatter matrices for fail and success classes
        # scatter_matrix_fail = utils.get_scatter_matrix(dataset_f,
            ◦! mean_vector_fail)
        # scatter_matrix_success = utils.get_scatter_matrix(dataset_s,
            ◦! mean_vector_success)

        # # calculate the within class scatter matrix
        # S_w = (scatter_matrix_fail + scatter_matrix_success) / 2
        # S_w = np.array(S_w, dtype='float')

        # # set S_w to be a diagonal matrices: keep only the variances of
            ◦! electrodes, not covariances
        # S_w = np.diag(np.diag(S_w))

        # # calculate the pseudo inverse of S_w
        # S_w_inv = np.linalg.pinv(S_w)

        #self.w = np.dot(S_w_inv, mean_vector_fail - mean_vector_success)

        # calculate the weight vector
        self.w = (mean_vector_fail - mean_vector_success).values # no
            ◦! normalization

        # calculate the cutoff value
        self.cutoff = self.w.T.dot((mean_vector_fail + mean_vector_success)) /
            ◦! 2

    def predict(self, data_row: pd.DataFrame) -> list:
        """
        This method predicts the class given one datapoint
        """

        if self.w is None: raise Exception("LDA model not yet trained!")
        if "class" in data_row: data_row.drop("class", inplace=True)
        if "freqc" in data_row: data_row.drop("freqc", inplace=True)
        if "trial" in data_row: data_row.drop("trial", inplace=True)
        if "ptp" in data_row: data_row.drop("ptp", inplace=True)

        # calculate the row projection
        data_row = np.array(data_row)
        projection = data_row.dot(self.w)

        # predict
        if projection < self.cutoff:
            return 's' # classify as success
```

```
        else:
            return 'f' # classify as fail

    def test(self, test_set: pd.DataFrame) -> dict:

        """
        Given a test set this method tests the performance of the LDA model
        """

        y_true, y_pred = list(), list()

        for _, row in test_set.iterrows():
            y_true.append(row['class']) # save the true label
            y_pred.append(self.predict(row)) # save the predicted label

        results = dict()
        results["acc"] = metrics.accuracy_score(y_true, y_pred)
        results["b_acc"] = metrics.balanced_accuracy_score(y_true, y_pred)
        results["correct"] = metrics.accuracy_score(y_true, y_pred, normalize=
            ‹! False)
        results["incorrect"] = len(y_true) - results["correct"]
        results["n"] = len(y_true)
        results["w"] = self.w
        results["c"] = self.cutoff

        return results
```