DAMC Miniproject 2: Feature selection and CV for supervised learning

Théo Imler, Frédéric Freundler, Nicolas Freundler

November 19, 2017

1 Introduction

During an experiment, sixteen EEG channels are plugged on the participant, measuring ERP at 2400 different time points (Event-Related Potential). ERP are brain waves generated by external event. The participant has to follow a cursor and in 25% of the time, the cursor move in a unexpected manner in order to generate ERP. The sample is collected from 648 participant.

The purposes of the project are to select relevant features for splitting samples into to classes: expected and unexpected cursor movement; perform a thresholding for a one-feature classification and perform accuracy test on it, use `MATLAB` tools for classification and CV and evaluate them.

2 Methods

First we used `histogramm` tool to have a visual idea of the distribution of the feature value and gives a clue for finding relevant feature, i. e. with significantly different histograms when plotting the feature from class 0 with same from class 1. `Boxplot` summarized the distributions by plotting quartiles . The `ttest2` performed a Student test for two sets. Student test assumes the random variables in compared sets follow a normal distribution and have the same covariances. The statement "values from the two sets have the same expectation value" is called null hypothesis and is opposed to "values from set 1 have a different expectation value than those from set 2". Student test reject or accept zero hypothesis depending on p-value: the probability, if null H was/is true, to obtain something worse (more different set 1 and set 2). Below a critical p-value, null H is rejected. By default, `ttest2` use the 95% two-sided confidence interval, which leads to a critical p-value of 2.5%. The two compared sets are the samples of class 0 for some feature and samples of class 1 for the same feature. It would be a non-sense to cross features, because they come from different measurements; one would try to compare, let's say, a capacitance and a resistance. We separately applied the Student test to all features, although it's theoretically not a perfect idea. In fact we did not control whether the features are gaussian and have the same covariances, but it would be time-consuming to check all histograms. Features that gives an output `h=1` (null H rejected) were considered as relevant or significant for class separation. To build a useful model, we needed at least one significant feature.

Thresholding consisted to partition this 1-D feature space. When some points were at one side of the threshold, they were assigned to a class whereas the other ones were assigned to the second class. Histograms help to find an good location for the threshold. The threshold should be somewhere between the modes (maxima) of the histograms. We applied thresholding to the same feature and calculate classification error and class error as they are described below.

$$classification\,error = \frac{\#\,of\,misclassified\,samples}{total\,of\,samples}$$

$$class\,error = \frac{1}{2}\left(\frac{\#\,of\,misclassified\,samples\,of\,class\,A}{total\,samples\,of\,class\,A} + \frac{\#\,of\,misclassified\,samples\,of\,class\,B}{total\,samples\,of\,class\,B}\right)$$

Thresholding works only for one feature. If one takes at least two features with their thresholds, it's likely to have unclassifiable samples because thresholding of one feature will have returned "class A" whereas thresholding of the other one will "class B". Other methods are presented later to allow multiple-feature classification.

We then split the samples into a training set and a test set. The proportions of class should be maintained, otherwise it would change the classification error. Also, if we choose unwisely, we have a risk to have 100% of one class in the training test, and we'll be unable to build a classifier. The threshold is applied on the training set and the test set, and two pairs of metrics were calculated: training classification and class errors, test classification and class errors. We chose the metrics coming from the training set: it doesn't make sense to train a model on the test set instead the training set, because it is why this one is called a training set. However, once trained, we used the class error of the test set to assess the quality of our classifier.

Afterwards we trained `MatLab` built-in classifiers with the object `fitdiscr` with different hyperparameters. These object works in general with any number of features.

- Linear discriminant
- Linear diagonal discriminant

- Quadratic discriminant
- Quadratic diagonal discriminant

The discriminant is a threshold method. Linear discriminant assumption is that the covariance matrices of different classes are the same, whereas quadratic discriminant is used when one assumes that different classes have different covariance matrices. They result respectively in a straight line and a parabola for thresholds. Diagonal is an additional assumption, which is the features are uncorrelated, i. e. the covariance matrices are diagonal. We trained the model on the whole data set, and applied the classifier the same data set with function `predict` and obtained training errors. `fitdiscr` allows to change the prior probability that we could us if we wanted to assess more weight to a class if, for example, the proportion of labels in the training set was different in the population. By default, it takes the empirical densities; this is why conserving the proportion among the subsets is important. We re-ran the last step with a different prior probability.

We wanted to check the performance of these built-in classifiers. I. e., we took the testing errors into consideration. The data set was therefore split into two sets of same size. We calculated the different error metrics as previously described: training classification and class errors (`set1`-trained model on `set1`) and test errors(`set2`-trained model on `set1`), always several times with different hyperparameters (type of discriminant, specification of the prior probability). We repeated this operation swapping sets (`set2`-trained model on `set2` and `set1`-trained model on `set2`).

Cross validation (CV) is a way to automatize the performance assessment with an iterative method. The data is partitioned in several (disjoint) subsets. At each step, a different subset is chosen as the testing set whereas the remaining subsets will become the training set. This method is a good solution when the data isn't big enough. Indeed, with such data, one never knows how to have a good training (with a big training set) and a good testing at the same time. With little training data, the probability for a bias is high, and with little testing data, probability of having a test data that is far from its ideal distribution is high (and that will lead of the conclusion that the model is not accurate even if it was). `MatLab` offers a partition object called `cvpartition` in which we put three arguments. The first one is either the number of samples or the the label vector, the second one is the type, here `'kfold'`, and the last one is the subset number. When the sample and the subset numbers are equal, it is called a leave-one-out cross validation (LOOCV), and we wrote `'LeaveOut'` .

So far we took the classes as they were : categories. In applications, they can have more sense : class A possesses a quality which class B is devoid of. Medicine is a good example, there are a healthy patients that forms a class "control group" and patients who bare the disease and thus form the other class. The outcome of the classification is "negative" and "positive", meaning healthy and ill. So far we gave the same importance to misclassification (form A into B and reversed). Now a misclassified healthy patient as sick (false "+") does not have the same importance than a suffering patient as healthy (false "-"). For model selection we used 10-fold CV. Each classifier trained with a

different Nset is considered as a different model. To choose the best number of feature we ranked them using a filter method : Fisher scoring. For a single feature it calculates a score that measures intra-class similarity and inter-class dissimilarity :

$$S(x_i) = \frac{\sum_{j=1}^{c} n_j(\mu_{ij} - \mu_i)^2}{\sum_{j=1}^{c} n_j \sigma_{ij}^2}$$

Thus the features are ranked in function of their discriminative power. `'Matlab' rankfeat` function was used with Fisher method. In each fold, the `diaglinear` classifier was tested with single best feature, then two best features and so on. Each time, the training and testing errors were saved. The decision about the number of features was based on the average test error of all fold.

We could not use the same dataset to choose the best hyperparameters of our model and to test his performance without obtaining a biased error. In order to test the model on a unseen dataset we computed nested CV (fig.1): the classifier performance test will be done in an outer CV while the model selection will be performed in an inner CV (10-fold) inside of each fold of the outer CV. From the nested CV we obtained two errors per outer fold : the *validation error* computed for each hyperparameters value during the inner CV and the *test error* computed on a test fold of the outer CV. In each outer fold, the model with the lowest mean validation error across inner folds was selected : we measured the performance of the model that won in the inner fold on the outer testing test. Mean of test error across the outer folds gives an unbiased estimation of the model performance.

The filter method as Fisher-based algorithm only looks at individual discriminating power of features and do not take in account the possibility of inter-correlation between features which could lead to redundant information. In order to treat this possibility we chose to apply a wrapper method to our feature : forward features selection. This method selects the best performing single feature in term of classification error. Then all pairwise combinations of the remaining features with the already selected one are tested. The one with the best joint performance is selected and we add a third feature that leads to the best joint performance with the two features already selected, and so on. We performed it using `sequentialfs()` method : `[sel,hst]=sequentialfs(fun,features,labels,'cv',cp,'options',opt);` This `MATLAB` built-in function performs itself the inner-loop CV to determine the optimal number of features. We chose a 10-fold CV `cp`.

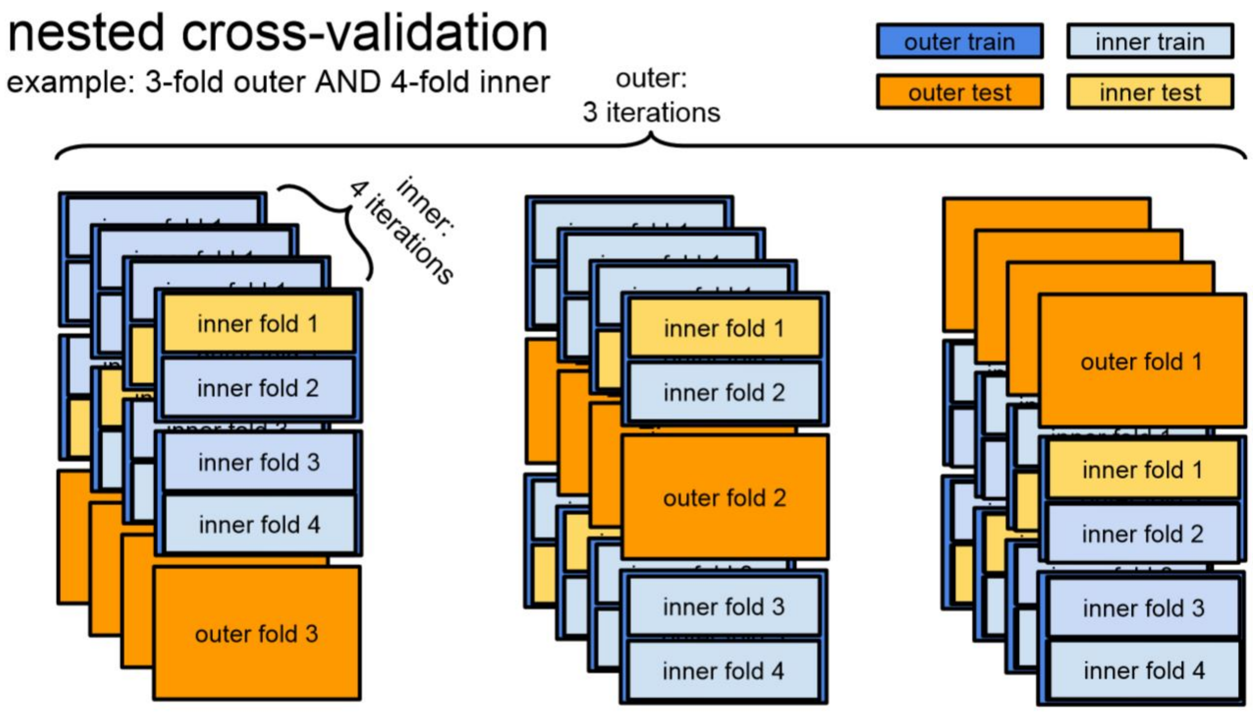

Figure 1: Mapping of a general nested CV process.

The function uses a specific criterion to evaluate the performance of a feature combination and stops adding feature when there is no more improvements in the prediction criterion.

The criterion was defined in the function `fun`, and corresponds to the validation error - in the inner loop. For each candidate feature subset, `sequentialfs` performs CV by repeatedly calling `fun` with different training and test subset of features and labels.

`sel` returns a logical vector indicating which features are finally chosen and `hst` contains a scalar structure with the criterion values computed at each step and a logical matrix indicating the feature selected at each step. In the end, we look for the minimum criterion - the optimal validation error.

Each time it is called, fun trains a model, then predicts value for the test subset using that model and finally stocks the validation error - the criterion - in `hst`. `sequentialfs` chooses the candidate feature subset that minimizes the mean criterion value. This process continues until adding more features does not decrease the criterion. Finally, we varied the classifier type as a hyperparameter on top of the number of features, running a nested CV. Features

selection - forward features selection and Fisher scoring - is one approach of reducing the dimensionality of our data. Another approach is features *extraction*. It's what we did by performing Principal Component Analysis (PCA). PCA is an unsupervised technique allowing us to project the high-dimensional samples space on a lower-dimensionality space that carries sufficient information. This is achieved by combining features.

Each sample x with d features can be represented as a linear combination of a set of d orthonormal vectors $u : x = \sum_{i=1}^{d} z_i u_i$, where $z_i = u_i^T x$. If we retained only $M < d$ of the vectors u, then we only need M coefficient z_i and replace the remaining coefficient by constant b_i. Thus each vector x is approximated by $x_{app} = \sum_{i=1}^{M} z_i u_i + \sum_{i=M+1}^{d} b_i u_i$. Thus we performed a dimensionality reduction since the original vector x with d dimensions is now represented by a new vector z with $M < d$ dimensions.This z vector represents the original features projected along the new vector space of vectors u_i.

Then the aim is to find the b_i and u_i that minimize the square error between the original data and the projection. Minimizing the square error with respect to the choice of the basis vectors u_i corresponds to satisfying $\sum u_i = \lambda_i u_i$ where$\sum$ represents the covariance matrix. Thus vectors u_i are the eigenvectors of the covariance matrix. So the minimum error is obtained by choosing the $d - M$ smallest eigenvalues and their corresponding eigenvectors, as the one to discard. Each of the eigenvector u_i is called a *principal component* or *PC*. Thus the original samples can be represented with a relatively high accuracy by projecting onto the first M eigenvectors.

We applied PCA to our features using `Matlab` function : `[coeff, score, variance]=pca(X)` which returns respectively the PCs the data projected on PCs and the variance of PCs.

This method calculates a new projection of our data set and the new axis are based on the standard deviation of our variables. So a variable with a high standard deviation will have a higher weight for the calculation of axis than a variable with a low standard deviation - PCA enhances the initial variance. Thus we normalized the data before the PCA using `zscore` in order that the PCA is not biased by a potential feature having a very higher variance comparing to the others.

To improve dimensionality reduction we applied PCA on the 100 most discriminative features using `rankfeat` beforehand and we chose to look at the variance of each PC and to keep the M PCs that represents 90 percent of the total variance.

PCA allows us to find, starting from a set of observations in a N-dimensional feature space, a M-dimensional space. The M PCs to be retained is an hyperparameter.

In order to not use correlated features we also performed another algorithm, using forward feature selection and classifier type selection inside a nested CV. The wrapper method was repeated inside each outer fold for each classifier type to choose the best features and the best classifier. The criterion of forward method was adapted for each type of classifier. We computed standard deviation and mean test error across outer folds.

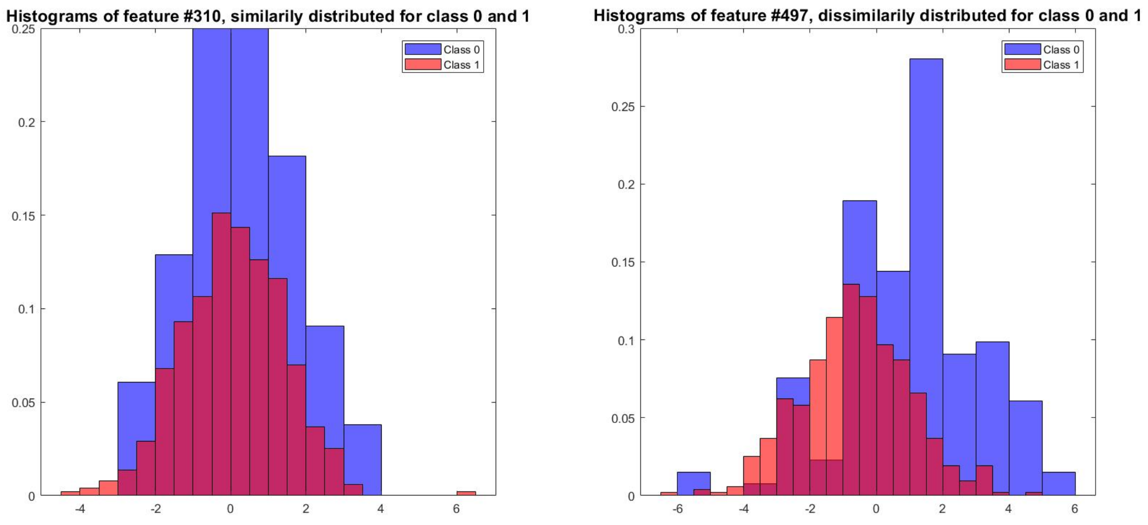

Figure 2: Histograms of class 0 and 1 for features 310 and 497. Modes for feature 310 are -1 to 1 and -0.5 to 0 in respect for classes, and modes for feature 497 are 1 to 2 and -0.1 to -0.5. We have such value because the software must have centered or nomalized them.

3 Results

3.1 Feature thresholding

3.1.1 Features comparison

We chose feature 497 as relevant and 310 as irrelevant, the histograms (see fig.2) show that the two classes have significantly different means and modes for 497 and close modes for 310. It is not enough to compare modes, we therefore compared features with boxplots (fig.3).

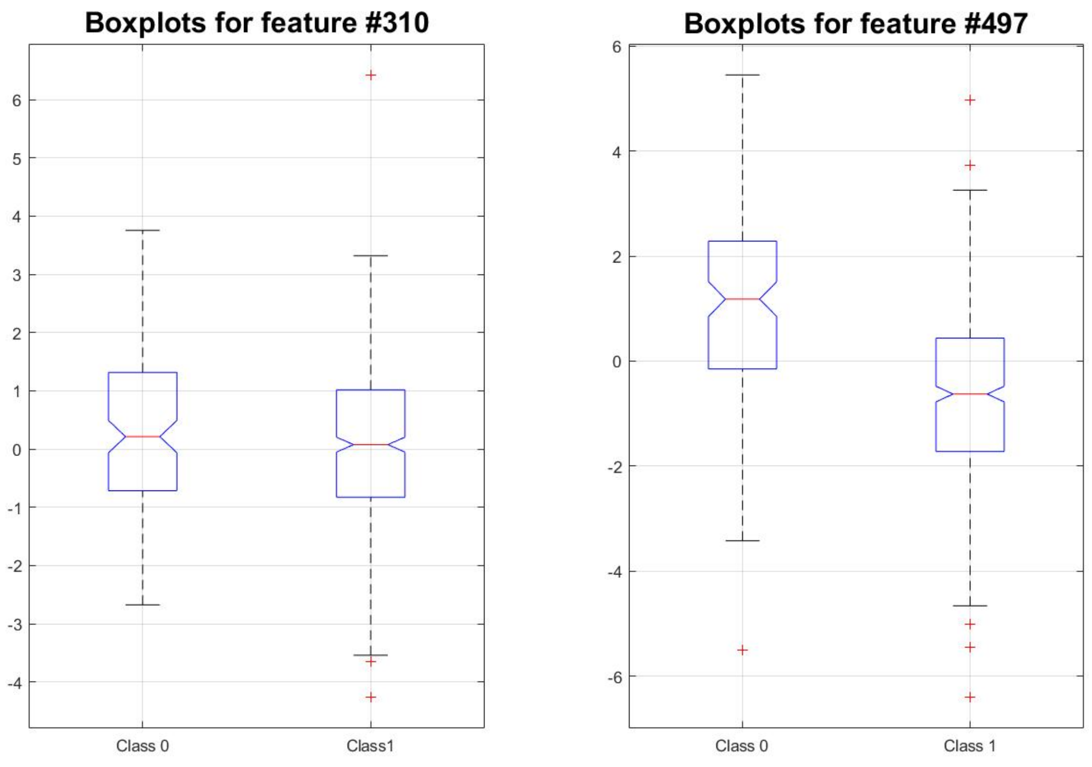

Figure 3: Boxplot with Notch heights of classes 0 and 2 for features 310 and 497. Red line are medians, top and bottom blue horizontal box limits are respectively the first and the third quartiles. Black lines are whiskers with maximum length of 1.5 interquartile (heights of blue rectangle). Notch heights are represented with the concavity at the median line, this height is the 95% confidence interval for median,

We see that the medians are more different for 497 than for 310. More interestingly, the notch heights (median 95% CI) of the two classes overlap for feature 310, the hypothesis "they have the same median" is not rejected, whereas it is rejected for feature 497. Presence of outliers can indicate that distributions are not perfectly normal. The selected features show some outliers in a low number: 9. We took them as measurement artifacts. The normality of the distribution is an important assumption for student test. The outcomes of the student test `ttest2` gave similar results for the means. For feature 310, the null hypothesis is not rejected and the p-value is 0.1107. For feature 497, the null hypothesis is rejected with p-value 3.1027E-20. The test reject null hypothesis for p-value inferior to 2.5%.

3.1.2 Thresholding

To begin, we use a naive threshold for feature 497: middle point between the means of the two different classes, 0.1704 (fig.4).

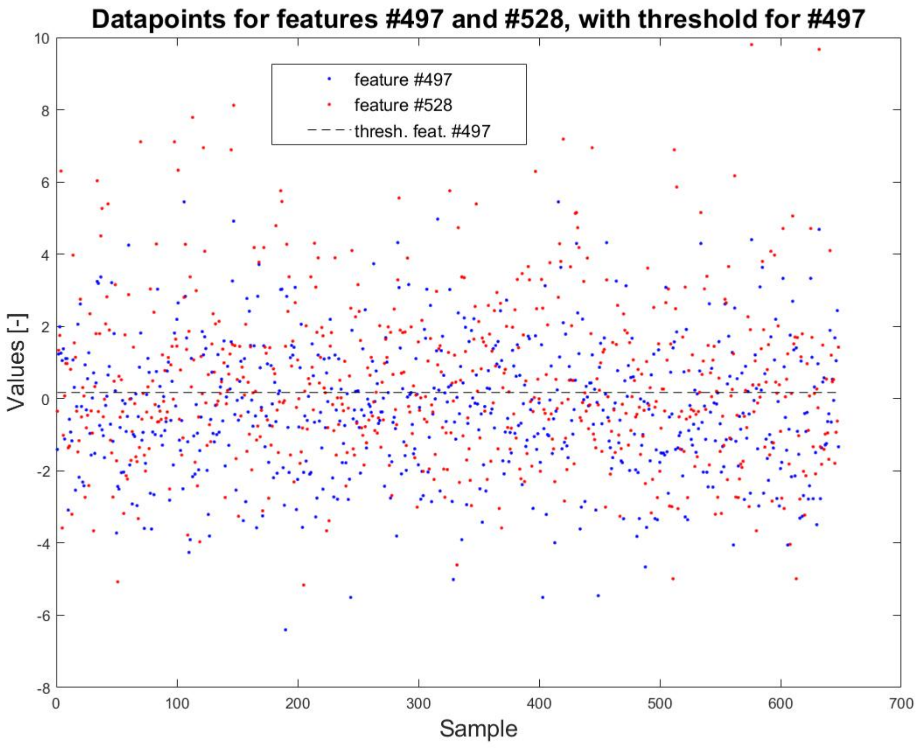

Figure 4: Representation of data for features 497 and 528, with threshold at 0.1704

The thresholding works only for one feature. We choose feature 528 as a second discriminative feature (student test p-value 1.0867E-38). Features 497 and 528 are differently distributed and it is therefore absurd to apply the threshold of 497 to 528. The classification error of our naive threshold is 30.7099% and the class error 31.9679%. For feature 528, the naive threshold is 1.2170 with classification and class errors 23.6111% and 26.3830%. The fact that the discriminative power of 528 seems better than 497 (lower errors) can be linked with the fact that its p-value is by far lower (order E-38 against E-20 for 497). A simple possibility is to draw a line as a function of multiple features (see fig.5). Not only does this approach visibly misclassify lot of data, but it is difficult to conceptualize inn more than 2 dimensions. Let's go back to feature 497. We slid the threshold value to see how errors evolves.

The difference between classification error and class error comes from the fact that classification error takes in account the proportion of classes. In our case, class 1 is overrepresented: 79.63%. We find this value in the classification error in fig.6: at extremely low threshold value, all class 1 are misclassified. Class error, on the other hand, ignores proportions. If we ignore prior probability, class error should be the metric to evaluate.

3.1.3 Generalization

We split the data into a training set and a test set. As we evaluated the threshold with class error, proportion of class in the new sets doesn't matter. However, we conserved it because we also wanted to compare classification error, for which proportions matter. For our training set,the lowest class error is 46.9168%, corresponding to thresholds 1.4735, 1.4918, 1.5101. We arbitrarily choose the first threshold 1.4735. The equivalent classification error is 26.2346%.

We applied the thresholding to the test set. The classification error and the class error were 65.4321% and 76.6032%. The training class error was already bad (49.9128%). We conclude that 497 is not as good as we thought for thresholding

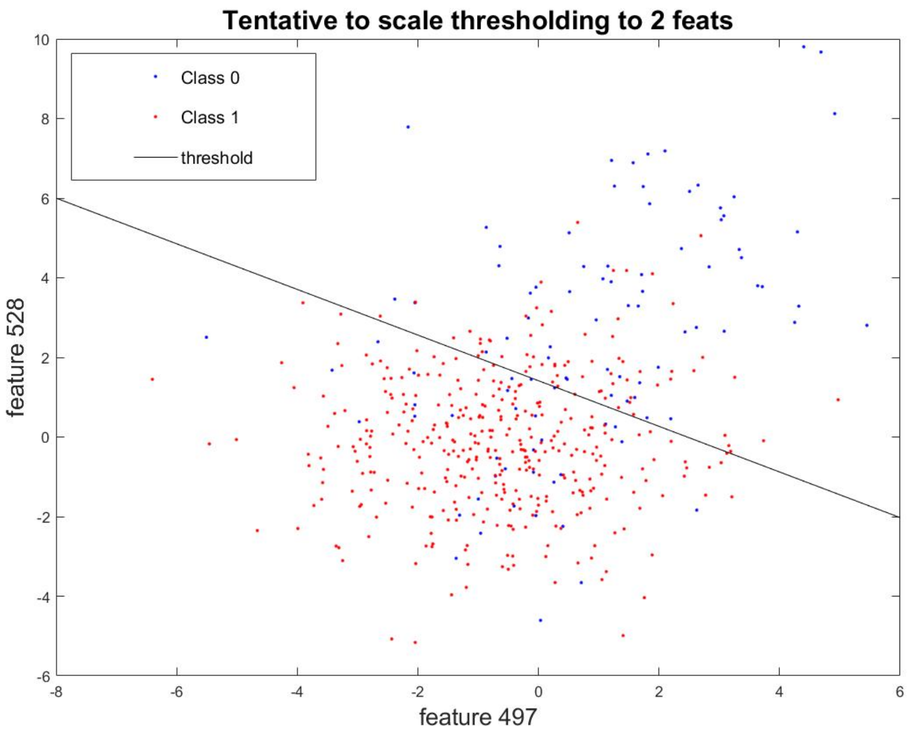

Figure 5: Slope of dark line -0.6857 and its intercept 1.5143.

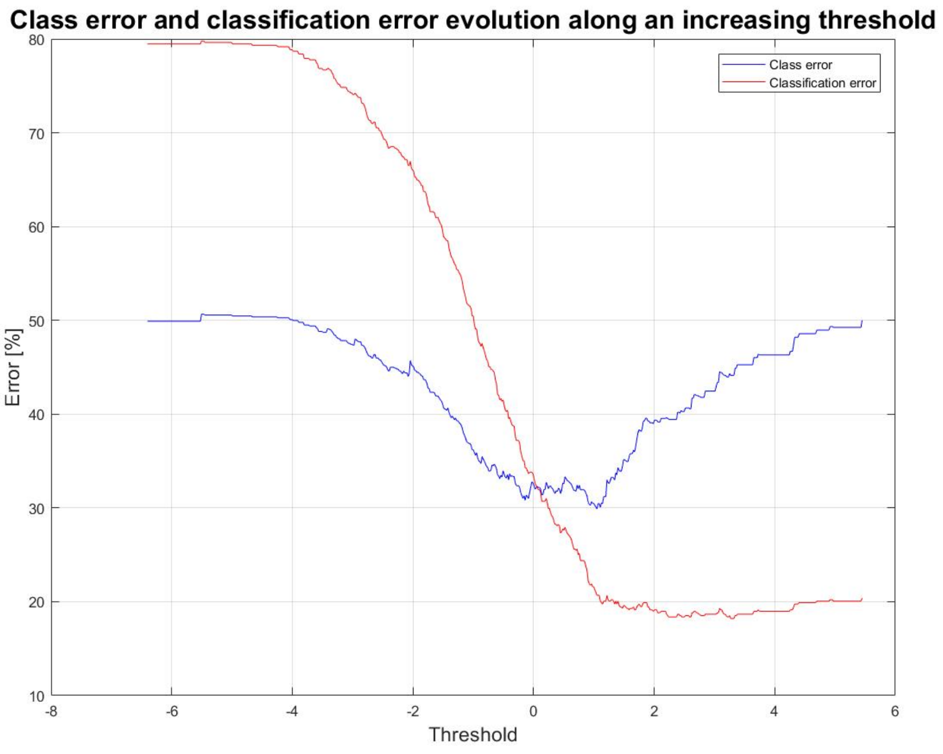

Figure 6: Classification error forms a sigmoid from approximately 80% to 20%. Class error shows a inverted-bell shape curve going from approx. 50% to 30%

3.2 Classifiers performance

The performance of the `MATLAB` built-in classifiers was tested on two different sets (*set*1 and *set*2). The classifiers were trained on the training set, thus training error does not allow to clearly address the performance of our model because of over-fitting. As we can observe on 7, the training error is always smaller than the testing error. It is explained by the fact that the prediction corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably. It is exactly what we observed for the linear classifier, training error was null meaning that our classifier is able to find a threshold which discriminates the different labels perfectly in the training set but the model is over-fitted leading to the biggest testing error compared to the others models. Performance was then assessed with the testing error. It is the `diagquadratic` classifier which was the most effective of our three choices. The repartition of the sets as training and testing confirmed this hierarchy and the stability of the performance. Thus it's our second most complex model (number of parameters for a diagquadratic classifier is equal to the number of class C multiplied by the number of features N. Diaglinear only takes N parameters whereas Linear N times (N+1)/2 parameters corresponding to variances and covariances) which had the highest performance (classification error equal to 0.12). Note that we could not train a quadratic classifier because the different classes shared common covariance for certain features leading to the impossibility to discriminate them using parabolic threshold.

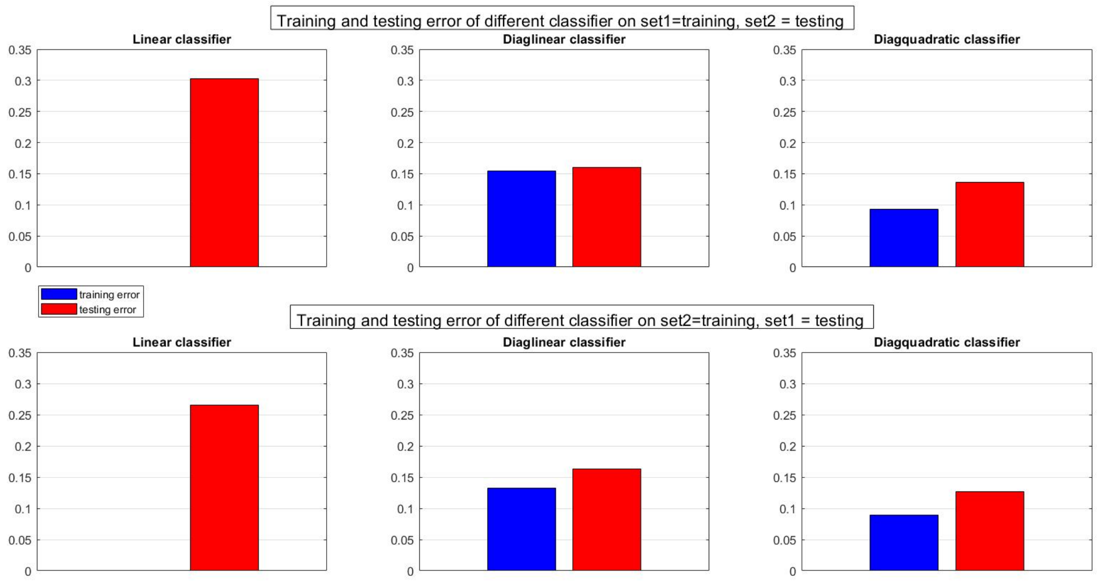

Figure 7: Performance of different classifiers on two different set : *set*1 was first used as the training set (*set*2 as test set, **TOP** graphs), and then the other way around (**BOTTOM** graphs). Training error (predicted on the set of samples that are used to train a classifier) and test error(predict on an unseen dataset) are indicated for each model of classifier. The `diagquadratic` classifier clearly stood out, yielding the smallest error. The `linear` classifier yields a null training error since it was trained on a subset of 324 samples which allowed it to discriminate perfectly the samples. The `quadratic` classifier could not be used, for at least two covariances are the same, hence preventing the matrix to be invertible, which is mandatory for quadratic discriminant analysis.

The separation of the dataset into training and testing set reduces the amount of data used for training so we used CV to be able to train a model on all samples. 10-fold and LOOCV were picked for estimating the classification accuracy. As we can observe in 8, the `diagquadratic` classifier was the best performing choice. Each of the two CV was performed two times using different repartition of our samples in the fold to be able to assess more effectively the performance of our model. Classification performance was similar and, logically, the error didn't change in the case of the LOOCV since each of the sample is used once a test set.

		Prediction outcome		
		Class 0	Class 1	total
Actual value	Class 0	54	29	83
	Class 1	12	229	241
	total	66	258	

Table 1: **Confusion matrix for the `diaglinear` classifier**: the predictive value for class 0 - proportion of samples truly belonging to class 0 amongst those having been predicted as being class 0 - is 81.8% and the predictive value for class 1 is 88.7%. However, the sensitivity for class 0 - the proportion of samples being predicted as class 0 compared to the total number of samples truly belonging to class 0 - is of only 65%, and the class 1 sensitivity is 95%. All in all, this means that the predictions of this classifier are robust (81.8% and 88.7%) but that the classifier misses a significant lot of class 0 samples in its classification (65% sensitivity).

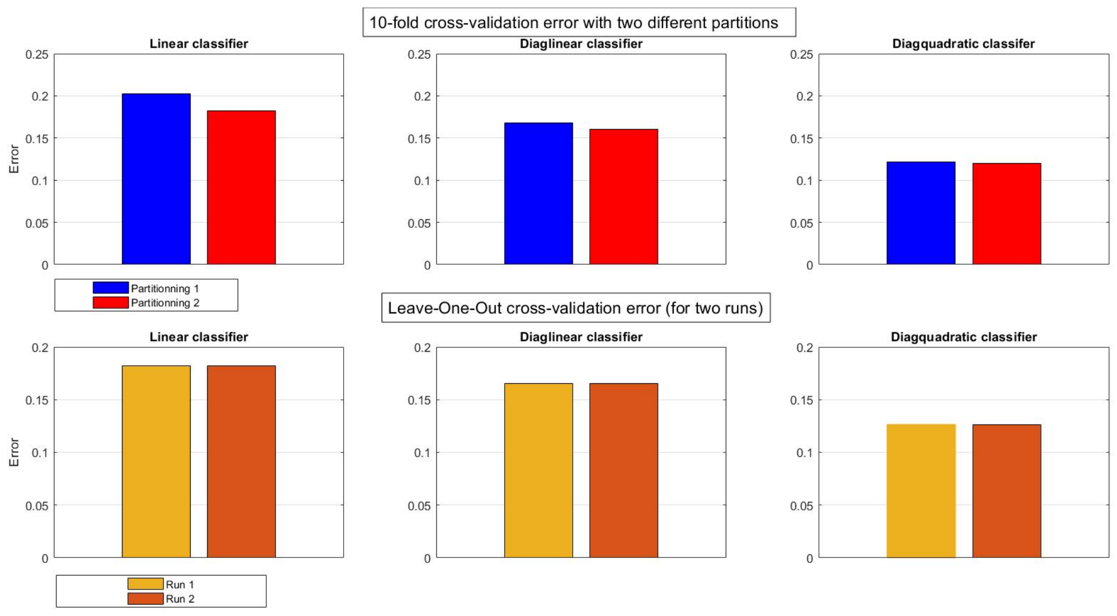

Figure 8: CV-error was calculated as the mean of the test error obtained over each of the fold (10 or 648)

About the efficacy of one method compared to the others, one has to keep in mind that when we perform LOOCV, we are averaging the outputs of 648 fitted models, each of which is trained on an almost identical set of observations (the 647 remaining samples); therefore, these outputs are highly correlated with each other. In contrast, when we perform 10-fold CV, we are averaging the outputs of 10 fitted models that are less correlated with each other, since the overlap between the training sets in each model is smaller. Since the mean of many highly correlated quantities has higher variance than does the mean of many quantities that are not as highly correlated, test error estimate resulting from LOOCV tends to have higher variance and smaller bias than does the test error estimate resulting from 10-fold CV. To get further insight into the performance of our classifier we studied the confusion matrix resulting from the original class repartition of our samples and the one of our prediction. Classification accuracy alone can be misleading if we have an unequal number of sample in each class as here. Furthermore the classification error can be easily recovered from the confusion matrix by dividing the sum of the false negative and false positive (respectively the class 1 sample predicted as class 0 and the number of class1 sample predicted as class 0) by the number of samples used in the classifier. The results in table 1 show that our classifier performed well, even if his sensitivity (true positive rate) was lower than his specificity (false negative rate).

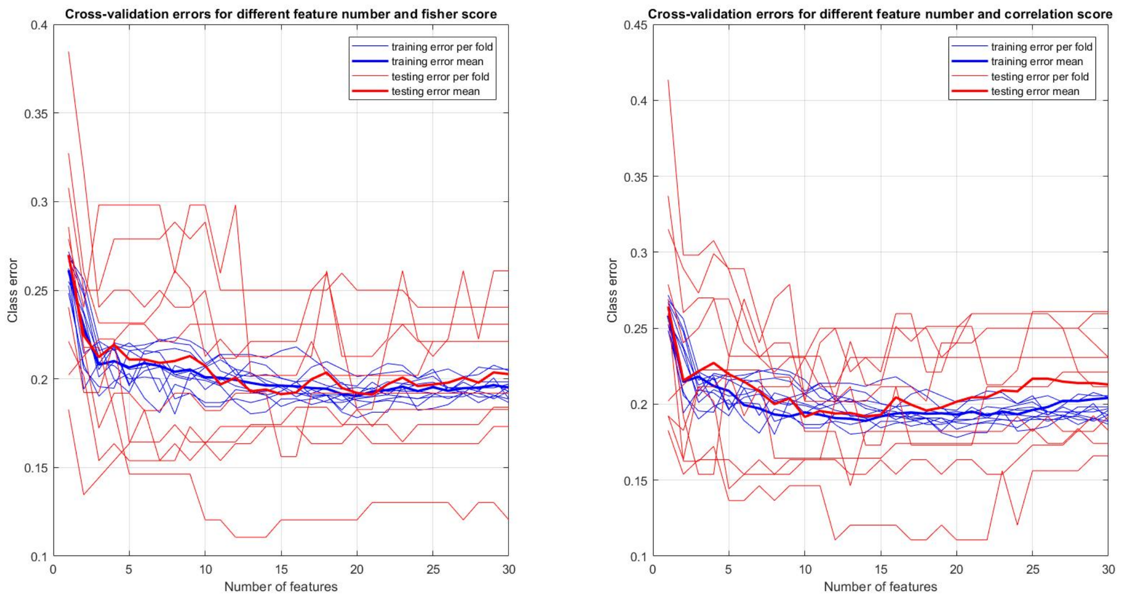

Figure 9: **Feature selection** either based on a `Fisher` (**LEFT**) of `correlation` (**RIGHT**) method. Thinner curves represent evolution of errors for each fold. Thicker ones represent means across outer folds. Minimum of thick red line, mean testing error, was used to retrieve the best number of features. The number of N_{sel} features selected with the `Fisher` criterion is 21 with a test error of ~19%, and for the `correlation` criterion it was 10 features with the same test error.

3.3 Hyperparameters optimisation

3.3.1 Simple cross validation

We then performed several CVs to select a best model based on feature ranking, choosing up to a maximum of N_{sel} selected features. We always varied the number of ranked features we used. The two first ones used the same discriminant classifier, i. e. `diaglinear`, but the ranking method varied: `Fisher, Correlation`. The number of subsets was 10. The `Fisher` method kept the first 21 ranked features with a minimal mean test error of 19% and the `correlation` method kept the 10 first ranked features with a test error of 19% (errors values are approximative because we saved figure 9 without saving numerical values). In comparison, a random classifier ran a thousand times yielded a mean test error of around 40% (data not shown), significantly higher than that of both previously tested methods. The behavior of training errors and test error is observable in 9 for both ranking method. The mean training and test errors behave similarly. but test errors per fold shows more variability than training error. This is expectable when data show themselves variability and test sets are small. `Fisher` and `correlation` have different outcomes, and this is because they calculate different score. Fisher score was already described, and correlation score calculates the score of a linear regression where classes are response variable. The problem with `correlation` is that it will detect a significant relation only if it is linear, we therefore used `fisher` for next steps.

3.3.2 Nested cross validation

To tune the model hyperparameter and test the model performance we computed the nested CV. In the light of our previous trials, we fixed the feature selection method using a `Fisher` criterion and used a `diaglinear` classifier for the moment. Using 3 outer folds, we obtained 18, 17 and 21 possible N_{sel} features selected, with a mean test error across folds of 19.95%. Errors are alway class errors. As we can see in fig.10, validation errors are generally lower than test and training errors, because it was the metrics used to pick the best model in the outer folds. Training errors show less variability than validation and test errors as it is expectable when data shows variability. Test errors are unbiased because outer test sets perfectly simulate new data, and it is expectable that they are higher than validation errors and more variable. Notice that there is no outliers, letting us presume that the errors are normally distributed - this aspect will be assessed in the final steps, section 3.4.3.

As stated above, we decided to first fix the classifier as `diaglinear`. As a second hyperparameter, we wanted to assess which classifier would be the best between `linear`, `diaglinear` and `diagquadratic`. out of 10 outer folds, we obtained 7 times `diagquadratic`, 2 times `linear` and 1 time `diaglinear`. It suggested us that `diagquadratic` might be the best choice. Furthermore, note that the `linear` classifier selected the highest number of N_{sel} features - above 40, where the others selected less features (data not shown).

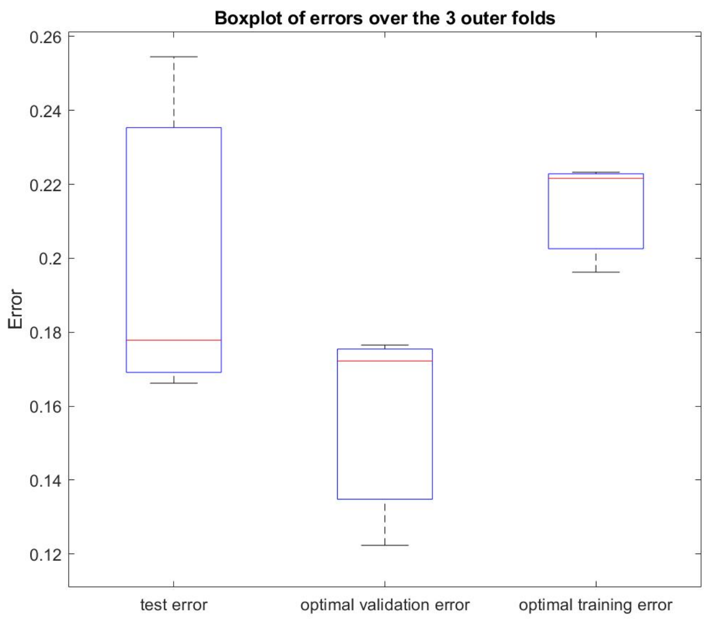

Figure 10: **Errors distributions** of our nested CV. As expected, the training error shows small variability and a relatively high error value since the inner sets on which it is computed is quite small and hence prone to misclassification. Concerning validation error - *inner* loop's test error -, variability increases as the error decreases, as expected considering a larger dataset. Finally, the test error across *outer* folds displays the highest variability and a higher error value compare to the mean of the two inner errors - which constitute the *outer* training error -, comforting us in the idea that an optimal model exhibits a test error slightly above the training error, with both around 20%.

3.4 Principal Component Analysis - PCA

3.4.1 First glance

We then looked into *Principal Components Analysis (PCA)* as explained in the Methods section. A quick look at the figure 11 confirms that not only does the PCA enhance the the variance of the principal components (PCs) regarding to that of the features, but that it also reduces the dimensionality (2400 original features for 647 principal components). It also classes the components from the biggest variance to the smallest. By deciding that we wanted to select the features that would represent 90% of the total variance, we retained the 44 first PCs (see fig. 12). We also wanted to implement this feature selection metric (PCA) in our previous nested CV process in order to appreciate the optimal number of N_{sel} selected PCs and compare it to the number of 44 of PCs representing 90% of the variance retained before. In order to get a more significant approach, we decided to compute and compare the mean error ratio (mean of $error_{training}$ over $error_{test}$), and isolating the N_{sel} selected PCs ensuring that the ratio stays between 0.86 and 0.8. This computation is displayed in figure 12 and the manner of choosing the ratio is explained in the legend. In short, this ratio ensures that the test error is slightly above the training error, so that we avoid over/underfitting. Of course, we checked that the errors are around 20% (result not shown). Normalisation was done *before* the PCA. Indeed, doing it *after* would be a non-sense since one of PCA's goals is to enhance the variances.

3.4.2 Enhancing the dimensionality reduction

After that, we wanted to reduce the feature space input in order to refine our feature selection and further reduce the dimensionality more using forward feature selection. As it is a very time-consuming method (it compares every next possible feature to the already selected feature(s)), we performed a previous ranking - after normalisation - in order to create a subset of 100 best discriminating features on which we applied the forward selection algorithm. The result is that only 10 features were kept by this wrapper method, in comparison to the 15 selected by ranking.

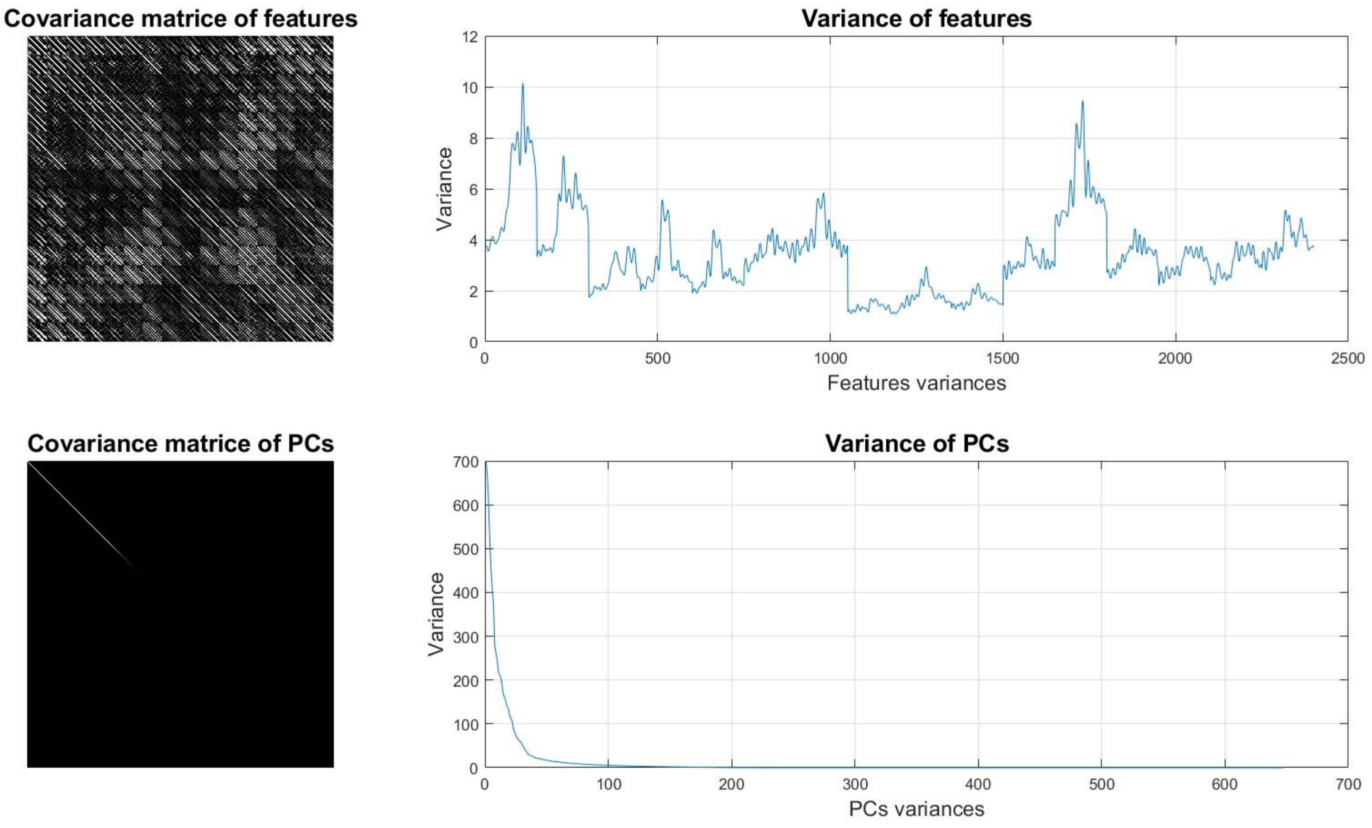

Figure 11: **TOP LEFT**: Covariance matrix of the original features space. Black indicates low covariance between two features and white a large covariance between them. The white diagonal elements indicates the self-explanatory correlation of every feature with itself, whereas the non diagonal elements, consisting of sparse white diagonals within visible "squares", suggests that some features are correlated by clusters, in what might be interpreted as the 16 electrodes in the experiment - indeed, we observe that the matrix is subdivided in 16 by 16 smallest squares. **TOP RIGHT**: This supposition about the electrodes-linked correlation between features is supported by the plot of the variance of the features: we observe 16 clusters features (each time approx. 100-150 features) that show significantly different variances, confirming correlation between features and the reducible dimensionality. **BOTTOM LEFT**: The covariance matrix of the principal components space, in opposite, shows no correlation whatsoever in between the non-diagonal elements, ensuring that the dimensionality is indeed reduced. Furthermore, the diagonal elements show a decreasing brightness, indicating that only the variances of the first N components have a true discriminating power. **BOTTOM RIGHT**: We observe that the PCA has reduced the dimensionality to the *P-1* space, where P is the number of observation - here, 648 samples. Furthermore, it has enhanced the the variance - the max variance is around 700, whereas it was around 10 for the original features space. The plot supports the assumption made on the covariance matrix that only the first N elements represent the quasi-totality of the variance.

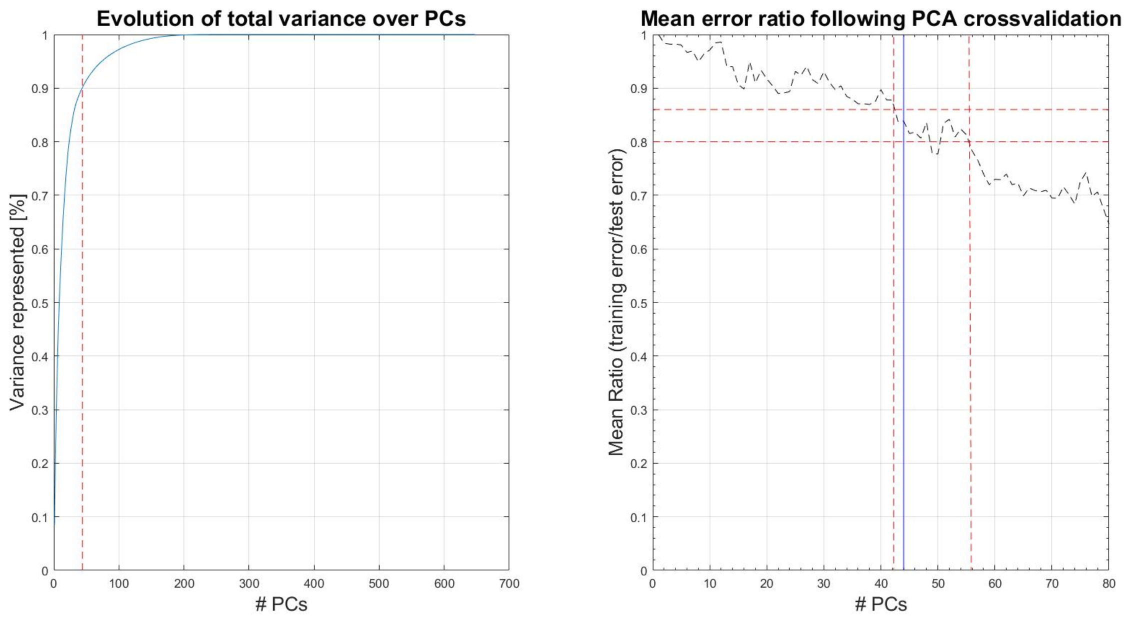

Figure 12: **LEFT**: Plot of the cumulative variance over the principal components outputted by the PCA. We observe that the 44 first PCs - red dashed-line - have enough discriminating power to represent 90% of the total variance over the PCs. **RIGHT**: Computation of the mean error ratio ($mean(\frac{error_{train}}{error_{test}})$) over the number of selected PCs when implementing PCA within the CV procedure. Here, the N_{sel} selected PCs is treated as a hyperparameter depending on the error fit: above 0.86, it implies that the training error is too high, meaning that the model is not good enough, whereas a value before 0.8 means that the training error is becoming too small and the test error too big, suggesting an overfitting. Of course, these values were set arbitrarily following our personal understanding of the notion.

At this stage, we started to see where combining different feature selection/extraction method and models could lead us. For instance, we could rank our features and then perform a PCA on the N features in order to reduce the input space.

3.4.3 Implementing a more complete nested CV

We also tried to implement the nested CV with forward feature selection - which performs the inner CV itself - and classifier selection on the outer loop. The mean test error across outer folds was 19.13% and the standard deviation 0.0523, which we deemed as precise enough. Across our 10 outer folds, the choice of classifier was stable for the `diagquadratic` classifier (8 out of 10, whereas the remaining 2 out of 10 choices were the `diaglinear` classifier), with a mean of 5 selected features (for both `diagquadratic` and `diaglinear`.

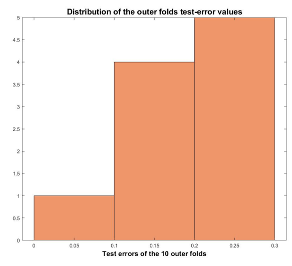

Figure 13: **Histogram** of the mean test error across the 10 outer folds.

Two features (from the original features space) appeared together 8/10 times, suggesting that they gather alone most part of the discriminating power. Interestingly, these features, when ranked by discriminating power (using `rankfeat`), appear not to be the top 3 features: the two features are feature 514 (rank 5) and 683 (rank 25). In 5/8 situations where they are associated together, they are also associated with feature 1729 (rank 1, best discriminating feature) or 1730 (rank 2).These 5 cases also correspond to the cases where the less features were needed (adding at most 2 features to the triplet, up to max. 5 features).

We quickly evaluated the statistical significance of the overall test error performances, which presented a mean equal to 19.13% and a standard deviation of 0.0523. A first t-test allowed us to reject the hypothesis that the test error values come from a random distribution (with mean = 50%), with a p-value of 1.67×10^{-11}. A Kolmogorov-Smirnov test on the standardised test error values did not reject the hypothesis that the data come from a standard normal distribution $\mathcal{N}(0, 1)$ (p-value = 0.7940), and a Wilcoxon-Mann-Whitney test did not reject the hypothesis that the test-errors distribution's median (and mean if it exists) is zero (p-value = 0.1543), even though the histogram of the mean test-error across the 10 outer folds is not shaped as a Gaussian (see fig.13): this results from the fact that a small sample of a normally distributed population possesses certain chances to be abnormally distributed. These statistics allow us to deem the test-errors distribution as normal and ensures us that we can apply the easiness of calculation and analysis specific to normal distributions.

3.5 Final Classifier: conceptualisation

In the light of what was performed in the scope of this project, we know that a lot of possibilities are offered to us in terms of dimensionality reduction - via feature selection (forward feature selection or ranking) and extraction (PCA) - and classifiers - `diaglinear`, `diagquadratic` , etc...

After a certain time of reflexion, we came up with a strategy to find the best model. Please note that the following is only *one possibility* amongst others. We would proceed by first splitting the dataset as desired in J-inner and K-outer folds, enabling it for nested CV. Then, we would normalise the training data set, filter them by ranking and choose the 100 most discriminative ones in order to do a supervised preliminary dimension reduction. We would then apply a PCA on this subset and perform a forward wrapping (such as forward feature selection) on the PCs. Of course we'd keep track of the μ (mean) and σ (standard deviation) - used to normalise the training set - and of the principal components in order to apply them on the validation and testing sets. From this inner CV, we would retain N_{sel} selected features. In order to avoid a too big complexity of the model, the discriminative criterion has been fixed as `diagquadratic`, since this classifier appeared most of the time in our previous simulations. We would finally establish the classifier's performance on the outer test set.

Note that the sole hyperparameter that we have is the number textit$\mathrm{N}_{\mathrm{sel}}$ of selected features, but we could have also deemed the model - whether we choose to use PCA or not, the feature selection method - and the classifier as hyperparameters, but that would increase the complexity of the algorithm.