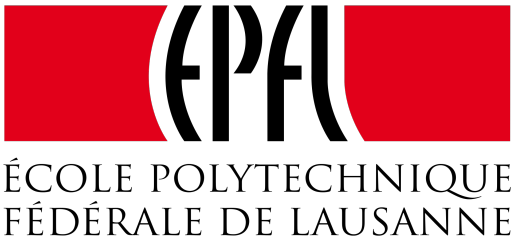
Ecole Polytechnique fédérale de Lausanne

Mini-project 2:
EE516: Data Analysis and Model Classification

Final Report

Constantin Krempp, Alexandre Delaux and Anastasia Bouzdine

November 19^{th} 2017



1 Introduction

The project consists in classifying data through supervised machine learning. The dataset represents Event Related Potentials (ERPs) which are electrical brain waves generated as reaction to certain external events. The experiments from which the data is extracted consisted for participants to watch a cursor moving towards an indicated target. In 25% of the time the cursor moved the opposite way unexpectedly, which elicited an error-related ERP in the participant.

More precisely, the dataset contains 648 samples corresponding to 648 observed cursor movements for one participant and 2400 features corresponding to the signal amplitude for sixteen EEG channels at different time points. For each of the samples, the label (correct or error-related ERP) is already known. This is why the learning will be supervised.

2 Methods

Each of the following methods are presented independently but their relevance and efficiency is generally highly dependent on how they are associated.

2.1 Classifiers

Choosing the right classifier is a crucial step in data analysis. We must choose an adapted classifier that minimizes the error without over-fitting and respects the initial distribution of the data.

2.1.1 Feature thresholding

The simplest way to assign a class to a sample according to a given feature is to choose a threshold and give a decision according to the position of the feature's value around that threshold.
In this approach after choosing the feature along which we want to classify the data, the goal is to find the optimal threshold.

Note: we here consider only one feature at a time, independently. The following classifiers are more powerful in the sense that they combine several features to produce the decision rule.

2.1.2 LDA/QDA Classifiers

Several types classifiers were considered:

- **Linear discriminant classifiers**
  These classifiers assume an equal covariance matrix for both classes. The covariance matrix can be either full, in which case the covariance between features is taken into account, or diagonal, in which case we make the hypothesis that all features are independent from each others. This method yields a linear separating hyperplane in both cases.

- **Quadratic discriminant classifiers**
  These classifiers assume that classes can have different covariance matrices. As before, the covariance matrices can be either full or diagonal. The separating hyperplane is then quadratic.

MATLAB functions *fitdiscr* and *predict* were used to train a classifier and give its prediction on the data.

To evaluate their raw quality on our data, we split it into two subsets of equal size. These subsets were created taking into account the distribution of our dataset between class 0 and class 1: it was made sure that both training set and testing set included the 20/80 ratio between class 0 and class 1. The benefit of splitting the dataset was that we could use one set to train the classifiers and test its performance on the unseen data of the other set.
For each classifier type, we computed the training error (training and testing on the same set) and the testing error (training on one set, testing on the other set) and we compared the results on both sets to assess repeatability.
A close attention was also paid to the prior distribution probabilities between the classes used by the classifiers and how they influence their performance.

2.2 Data exploration: Feature selection

Given the big amount of raw features representing our dataset, reducing the dimensionality of the working space and selecting the best features for discrimination are crucial steps before performing the classification task. The goal in this analysis is to assess features' potency (first independently and then jointly) of being efficient classifiers for the data: "good" features being the ones where each class tends to a different behavior whereas "bad" features are the ones where both classes present the same characteristic values.

To this end, several methods were introduced.
First, feature plotting and naïve approaches were used to categorize features ("good" against "bad"). Then statistical tools were employed to provide better insights on feature "quality". Eventually, more sophisticated approaches, such as Principal Component Analysis (PCA), Fisher classification and Forward Feature selection algorithm, were implemented to reduce the number of dimensions for the classifier and/or select the most discriminative features.

2.2.1 Naïve approach

Randomly plotting feature's distribution across classes is a good method to discover the dataset specificities but not very efficient to find relevant "good" and "bad" features.

An exhaustive naïve method has been implemented to easily detect the "worst" and "best" features of our dataset:
For each feature, the mean on each class was computed and subtracted from one another. The absolute value of these individual differences were then plotted for the 2400 features to be able to spot large differences and small differences, thus identifying candidates for "best" features and "worst" features.
Statistical distributions of the data across the candidates are eventually displayed for the spotted features to confirm the "quality" of the feature.

2.2.2 Statistical significance

For this study, the statistical *t-test* method has been employed.
For a given feature, the two-sample *t-test* tests datasets from two classes 0 and 1 for the **null hypothesis** that the distributions from both classes have equal means. The goal is to prove that this hypothesis can be rejected with a good confidence.

The *MATLAB* function *ttest2* performs the t-test and returns the decision and the corresponding *p-value*.

For a proper *t-test* usage, we have to assume that the samples come from normal distributions with equal covariances. Although normal distribution can reasonably be assumed, we noticed consequent divergence between covariances in our data. In that regard, this method remains quite approximative for our dataset.
T-test and *p-value* are also highly dependent on the number of samples in each dataset, and they especially need to be equal to provide meaningful results. As a consequence, when *t-tests* were performed, the dataset corresponding to class 1 was divided into 4 subsets with the same size as the class 0 dataset.

Practically, class 1 samples were divided into 4 determined subsets and a *t-test* was computed for each of these subsets against the class 0 samples.
A naïve feature selection upon statistical significance was then computed following these steps:

1. Starting with all features, only the ones rejecting the null hypothesis for the 4 tests were chosen.

2. Then, a statistical distribution of the *p-values* across the remaining features is computed for each subset. In each case we computed the list of the features among the first quantile (*e.g.* the 1% lowest *p-values*).

3. Finally, the 4 lists were fused to keep only the features appearing in each list. The resulting features were kept as the best features according to this selection rule.

2.2.3 Filter/Wrapper methods

Different methods can be applied for feature selection. Filter methods, such as Fisher score, measure the relevance of features by their correlation with dependent variable while wrapper methods measure the "usefulness" of a subset of feature by actually training a model on it.

We first implemented the cross-validation with Fisher-based feature selection using *rankfeat* method in *MATLAB*. In the nested cross-validation algorithm, this model requires the optimization of the number N_{Fisher} of features retained as an hyperparameter.(see section 2.3.2).

However, the drawback of this method is that it only looks at the individual discriminability of features, regardless their correlation. In this purpose, we also used a wrapper method, the forward feature selection (FFS) implemented through *sequentialfs* from *MATLAB*. It selects the best performing single feature and then tries all pairwise combinations of the remaining features with the already selected one, and returns the best subset of feature when a criterion of non-significant improvement is reached.

2.2.4 Principal Component Analysis

A very efficient way to select the most relevant features is to proceed to Principal Component Analysis, in which linear combinations of the original features will generate a new set of orthonormal vectors. These new features are uncorrelated and are ordered to maximize the variance of the data's projection. This enables to take off the redundancy among the features. One can then easily select the new features along which the projection of the data is the most dispersed. However this method doesn't take into account the class belonging of the data and therefore is not fully relevant to efficiently separate the data in their right classes.

The first step is to normalize the data, so that the features are comparable with each other in terms of variance. A PCA algorithm is then applied (*pca* function from *MATLAB* toolbox) and the N_{PCA} features with highest variance are retained. One solution is to select the N_{PCA} features that represent a predefined percentage m of the total variance of the data. Inside a nested cross-validation algorithm, this will be considered as an hyperparameter: several values for N_{PCA} are tested and we optimize the best value such that the validation error is minimized across inner cross-validation loops (see section 2.3.2 for further details).

2.3 Cross-validation methods

Simple cross validation is required to select the parameters for a specific model and to evaluate the performance of this model regarding the classification error. However, simple cross-validation does not permit to compare different hyperparameter values of the model used and can present a biased estimation of the true error.
This is why we use also nested-cross validation to evaluate the best hyperparameter value for our model and assess its robustness through statistical analysis. This also provides a more accurate estimation of the test error by introducing a new unseen dataset.

2.3.1 Cross-validation for parameter optimization

The data set is composed of 648 samples, which is not big enough to simply split it into a training and a testing set. Therefore, to select the parameters of a given model in an optimal way, we performed k-fold cross-validation. The dataset is split into k subsets, $k-1$ of them are grouped in order to build the training set, and the remaining subset is used for testing. All k combinations of training sets / test sets are iteratively realized, each of them leading to an error estimation. The final error value is defined as the mean of all the k estimations.

In our case, a 10-fold and a "leave one out" cross validation were realized, both using Fisher method for the number of features selected. The "leave one out" means that the subsets are composed of one only sample, the testing set is thus reduced to one sample. we used the *cvpartition* function of *MATLAB* to perform this partition labels were provided to the function so that each partition of the data would respect the 20/80 ratio between the two classes.

2.3.2 Nested cross-validation for performance estimation

Although our simple cross-validation loop allowed us to compare different sets of hyperparameters for our model (*e.g.* number of features), we are still unsure about the true performance of our model, and we need a new unseen dataset to test and to accurately estimate the test error. To do this optimally on a small dataset, we will do an outer k_{out}-fold cross-validation to test the performance of an hyperparameter or a combination of hyperparameters (*e.g.* a classifier, a number of selected features) while the model selection will be performed in an inner cross-validation inside each fold of the outer cross-validation.

In the inner cross-validation loop, we run k_{in}-fold cross-validation for every available model, *e.g.* combination of hyperparameters, to select the best model upon the mean test error across all inner partition (validation error). In the outer cross-validation loop, we measure the performance of the model that was selected, on an unseen separated external fold. In each outer cross-validation fold, we select the best model and compute the corresponding test error, and finally choose the "best" overall model by looking how frequently a given combination of hyperparameters was selected.

In this project, we tried different combinations : first we focused only on the number of features to select (Fisher method), then we added the choice of the classifier on top of the number of features, and tried different combinations along with a PCA to reduce the dimensionality of our space.

2.3.3 Statistical significance for Nested Cross Validation

To verify that the nested *CV* results are statistically reliable, we retrieved the overall test error performances after each nested *CV* we did and analyzed them statistically (see section 3.3.2). We can use the Student's t-test presented above 2.2.2 if the distribution is normally distributed, or a non-parametric statistical test, such as Wilcoxon signed rank test otherwise. The *MATLAB* function *signrank* already implements this non-parametric statistical test.

To decide whether to use a Student t-test or a non parametric test, we must first look at the normality of the distribution of test errors. To test it, we can perform a Kolmogorov-Smirnov test, implemented in *MATLAB* function *kstest*.

2.4 Error quantification

When trying to quantify the error made by a classifier on our dataset, different errors types can be considered.

2.4.1 Type of error

When comparing the labels from the classifier to the true labels, we can compute the classification error and/or the class error. On one hand the class error takes separately into account the influence of errors in class A and class B, thus reporting the prior repartition of the labels among class A and class B. On the other hand, the classification error uses the averaged number of misclassified samples (misclassifications on class 0 and class 1 are mixed up).

Both error types were either computed manually or outsourced to the functions *classerror.m* and *classification_ error.m* implemented ourselves.

2.4.2 Confusion Matrix

The confusion matrix of a classifier presents its raw performance *i.e.* the exact number of samples falling into the different categories issued from the comparison between the predicted labels to the true labels. The element in the first row and column is called **True Positive**, which represents the amount of correctly predicted class 0. Second column of first row is **False Negative** that shows the number of samples misclassified as class 1. For second row, first column is **False Positive**, the number of samples misclassified as class 0. The second column is **True Negative**, number of samples correctly predicted as class 1.

The confusion matrix was computed with the *confusionmat* command in *MATLAB*.

3 Results

3.1 Classifiers

3.1.1 Feature thresholding

As explained in section 2.1.1, a basic classifier may be implemented with a threshold on a given feature. The task for the learning algorithm is then to optimize the threshold value to get the least error rate. We here test the algorithm on the whole dataset. To achieve that, we represented the ideal normal distributions of each classes according to the within-class mean and variance. The reference threshold (or 'unbiased threshold' in Figure 1) was chosen to be where those normal distributions intersect each other. Then we explored different thresholds around this reference, trying to minimize the error rate. However, as you see on Figure 1, this is highly dependent on the error type we choose to compute.

Considering the classification error (Figure 1 *left*), you notice a strong bias between classes because their incidence rate differs in the original dataset. When displacing it towards the mean of class 0 distribution (to the left in Figure 1), you will get more errors for class 0 samples but less for class 1 samples. As there are less samples in class 0, the error rate will start by decreasing because of the overwhelming importance of class 1 over class 0. At some point it will increase again because you stop winning accuracy on class 1 samples. The opposite behavior occurs when the threshold moves towards the mean of class 1: the error rate is immediately skyrocketed.
The class error (Figure 1 *right*), by giving the same weight to each class, shows a symmetrical behavior around the 'unbiased' threshold. Moving away from this value automatically increases the error rate because the algorithm will always miss more samples from one class than the number of hits he will gain on the other class...

With those remarks in mind, it is easy to understand why the optimal threshold is the unbiased threshold when considering the class error and differs from it (biased towards class 0 mean) when considering the classification error (as shown on Figure 1).
This enlightens the importance of choosing well the error type, according to the initial distribution of the data and the goal pursued with the learning task.

In this project, we decided to mainly focus on the classification error instead of the class error, as it appears to better fit our model and its unbalanced repartition of samples' classes. Indeed, balancing the errors of our repartition would somehow misrepresent the real dataset, which has only 20% of chances to be labeled 0.

See code worksheet *Project02_threshold_2.m*.

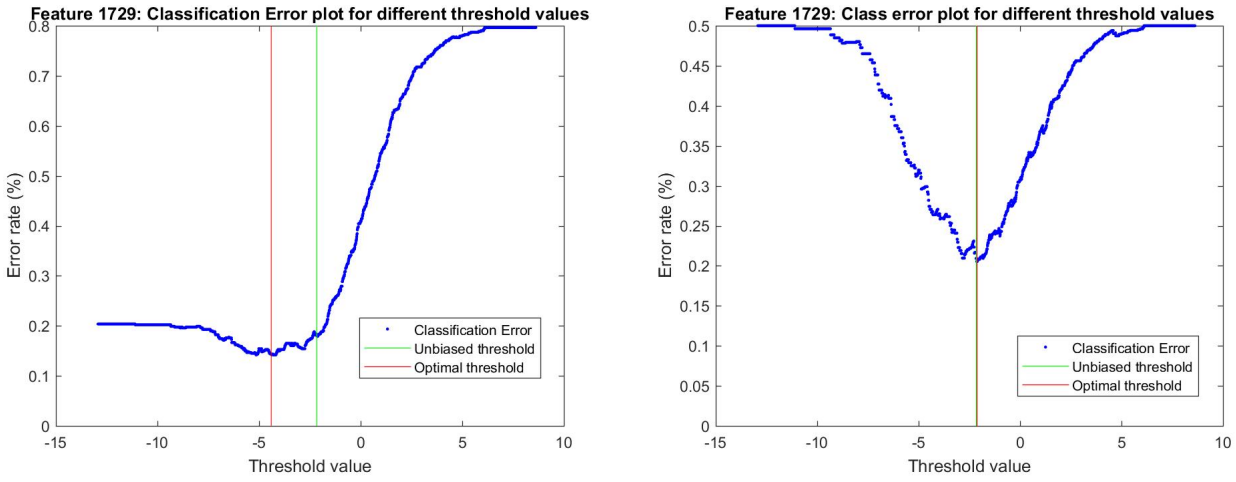


Figure 1: Feature 1729 example of error rate evolution according to threshold definition: comparison between Classification Error (left) and Class Error (right)

3.1.2 LDA/QDA Classifiers

We computed both classification error and class error for each type of classifier, with or without uniform prior, performing the training and testing on the same or different set. Given the high number of possibilities that it represents, this step helped us to choose the "best" classifier according to us, that we would preferentially use further in our project (unless explicit comparison between classifiers is mentioned). We based our analysis on Figure 2.

	Model	Linear		diagLinear		Quadratic		diagQuadratic	
	Uniform Prior	without	with	without	with	without	with	without	with
Training Set 1 Testing Set 1	Classification Error	0,09876543	0,15123457	*0,13271605*	*0,22530864*	**0,012345679**	**0,03395062**	0,11419753	0,17901235
	Class Error	0,19168428	0,16261452	*0,22427766*	*0,25422833*	**0,01902748**	**0,03258338**	0,17882311	0,196969697
Training Set 2 Testing Set 1	Classification Error	**0,12037037**	0,17592593	0,1882716	*0,25925926*	0,12962963	**0,13580247**	*0,19753086*	0,200617284
	Class Error	**0,20525018**	**0,20630728**	0,29298802	0,28118393	0,31254405	*0,28823115*	*0,31571529*	0,22181113
Difference Set1 (Testing-Training)	Classification Error	**0,02160494**	0,024691358	0,05555556	0,03395062	*0,11728395*	*0,10185185*	0,08333333	**0,02160494**
	Class Error	**0,01356589**	0,04369274	0,06871036	0,0269556	*0,293516561*	*0,25563777*	0,13689218	**0,02484144**
Training Set 2 Testing Set 2	Classification Error	0,11728395	0,12654321	*0,12654321*	*0,11209877*	**0,012345679**	**0,01851852**	0,09876543	0,13580247
	Class Error	0,225863284	0,1358351	*0,22603946*	*0,19326991*	**0,02466526**	**0,02854123**	0,19732206	0,15856237
Training Set 1 Testing Set 2	Classification Error	**0,14197531**	0,15740741	*0,15740741*	*0,2037037*	0,14506173	**0,14814815**	**0,14197531**	*0,2037037*
	Class Error	0,28083157	**0,19467936**	0,290521494	0,26321353	*0,33350951*	*0,318534179*	**0,25828048**	0,22938689
Difference Set2 (Testing-Training)	Classification Error	**0,024691358**	0,0308642	0,0308642	**0,02160494**	*0,13271605*	*0,12962963*	0,04320988	0,06790123
	Class Error	**0,05496829**	**0,05884426**	0,06448203	0,06994362	*0,30884426*	*0,28999295*	0,06095842	0,07082452

Figure 2: Computation of the training and testing error for each classifier, and the difference of these errors for both sets. In bold green are the best values - error minimization - and in italic red the high errors. Both error types were tested and *Uniform prior* option influence is also presented.

As exposed in section 3.1.1, we will focus only on the classification error (it is more respectful to the distribution of our data and gives lower error rates).

Uniform prior can be viewed as giving to the classifier an equal probability for every class we are trying to predict. Yet, in our problem, the distribution for labels 0 is 20% (*i.e.* a prior probability of 0.2), and setting the uniform prior for the positive cases to be 0.5 overcomes the imbalance effect of the original distribution and bias the repartition of our data. Indeed, when combined with classification error, we can see on table 2 that addition of *Uniform prior* option generally worsens the error rate. We will not use this option later.

An interesting information from Figure 2 is the difference between the Testing and Training error when changing the set for the training of data. This difference is quantifying how much our classifier might be overfitting. A low difference is expected for a "good" classifier, whereas a high difference - such as for quadratic classifier - makes us suspect that the classifier is actually overfitting. Eventually, only the case when the training and testing sets are different will be considered, as we want our classifier to be efficient for unseen data.

Combining these parameters - classification error, non *Uniform prior*, low difference Training/Testing error and performance on unseen data - the linear classifier appears to be the most effective classifier (most frequently represented in green in purple cases). This is why we will further focus only on linear classifier in the project, although we could have performed more iterations in order to assess this result with higher viability.

Using linear classifier, we computed the confusion matrix as a mean of all confusion matrices obtained out of 10-fold cross validation :

$$\begin{pmatrix} 78.99 & 53.01 \\ 28.56 & 487.94 \end{pmatrix}$$

As expected, **True Negative** is the highest (as we have more labels 1 than labels 0), followed by **True Positive**. Confirming what we saw with section 3.1.1, the classifier preferentially avoids errors on class 1 (**False positive**), degrading the number of **False Negative** errors.

See code worksheet *Project02_ Classifiers_ 1.m.*

3.2 Feature selection

3.2.1 Naïve approach

The naïve approach was implemented on all features at the same time to comparatively assess the 'quality' of each feature. We plotted the mean difference between features and from this graph it was easy to narrow down specific features according to their 'quality'.

Histograms and boxplots have been used to visualize the data distribution across classes and confirm in each case the previously assessed quality of the feature (see Figure 3 for examples).
We can notice here that for 'bad' features, there is a strong overlap between the distributions on the histogram and the boxplots have close means and boxes cover very similar ranges. On the contrary, for 'good' features, the overlap is minimized and boxplots are clearly separable from each others.

See code worksheet *Project02_ Naive.m.*

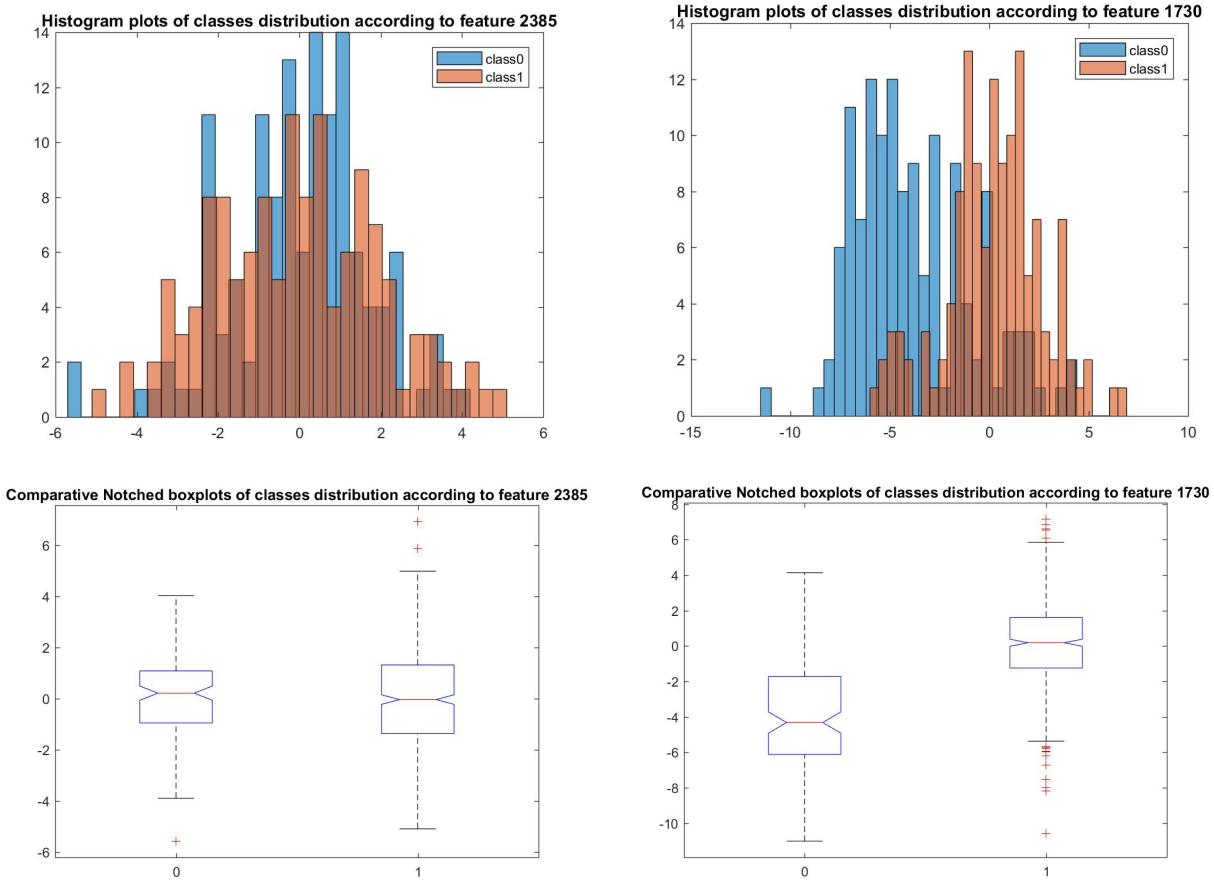


Figure 3: Histograms and boxplots for features found by the naïve approach: Example of a 'bad' feature (left) and a 'good' one (right)

3.2.2 Statistical significance

In order to find the best features for the classifying task, we used the procedure described in section 2.2.2.
All 2400 features were analyzed using the t-test. The code allows you to select different quantiles of the p-value distribution but we used 1% because we did not want too much features in the end. The output of this procedure gave us 4 features **(513;514;1728;1729)** that are thought to be independently able to split the data into the right classes minimizing errors.
However this does not gives us a performance-ordered list of the features (unlike *Fisher* method or *PCA*).

See code worksheet *Project02_ ttest.m.*

3.2.3 Filter/Wrapper methods

The Fisher ranking can first be performed on the whole dataset, to get an overview of the most discriminative features. It is interesting to note that out of the five best ranked features with

Fisher, we find the 4 features predicted by the naïve method from section 3.2.2, thus giving some credit to this method for finding good features.

We performed a nested cross-validation with the linear classifier for both Fisher method and Forward Feature selection (FFS): $k_{in} = 10\ folds$ and $k_{out} = 20\ folds$ in order to have enough statistical data.

In the case of Fisher score, the optimal number of features was found to be $N_{Fisher} = 5$ and this result remains quite stable over repetitions. The corresponding average Testing error is 0,101. In the case of Forward Feature selection (computed over the 2400 features), the optimal number of features was found to be $N_{FFS} = 5$ or $N_{FFS} = 4$ and this result varies over repetitions. The corresponding average Testing error is 0,103.

The boxplots presented on Figure 4 were calculated using data from folds where $N_{Fisher\ or\ FFS} = 5$, which accounts for the most representative case (14 out of 20 for Fisher score, and 6 out of 20 for Forward Feature Selection).

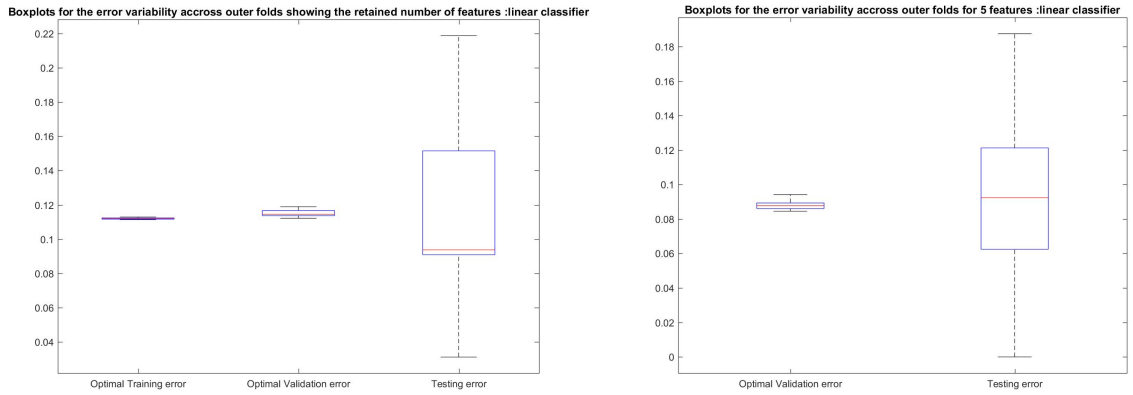


Figure 4: Boxplots of the statistical distribution of different errors for Fisher score (left) and Feature Forward selection (right) - the training error not being available in that case.

Finally, both results are close and the corresponding testing error are very similar, comforting us that the number of features to select should not exceed 5. We can notice that Fisher score method presented more stable results over repetitions and a lower standard deviation, making it a good choice for the model.

Statistical significance: For Fisher Score, mean Testing Error = 0,11 with a standard deviation of 0,05. T-test rejects the hypothesis of a distribution with mean 50% at the 5% significance level with a p-value equal to $1,7 \times 10^{-18}$. *For Forward Feature Selection, mean Testing Error = 0,10 with a standard deviation of 0,05. T-test rejects the hypothesis of a distribution with mean 50% at the 5% significance level with a p-value equal to* $4,2 \times 10^{-18}$. *Data was verified to be normally distributed by Kolmogorov-Smirnov test.*

See code worksheets *Project02_ Nested_ CV_ Fisher.m* and *Project02_ FFeature_ Selection_ Nested_ CV.m.*

3.2.4 Principal Component Analysis

As explained in the methods 2.2.4, different ways can be chosen to evaluate the number N_{PCA} of principal components (PCs) to keep. Actually, a relevant method is to make a first discrimination of the features using PCA, and then keeping the most relevant through a Fisher method or a forward feature selection. To get familiar with PCA, we first computed it on our whole dataset. As shown on figure 5 *left*, the explained cumulative variance quickly reaches a plateau thus giving hope for a considerable reduction of dimensionality using PCA transformation. For example to represent 90% of the total variance, one should select the 49 first PCs out of 2400.

However, such a method to select the number of PCs isn't very relevant, as it uses all the samples without letting a test set aside. The risk here is to overfit the model to our data. One should better split the data into a training and a testing set, that will be done in the cross-validation methods presented in the section 3.3.1.

See code worksheet *Project02_ PCA.m.*

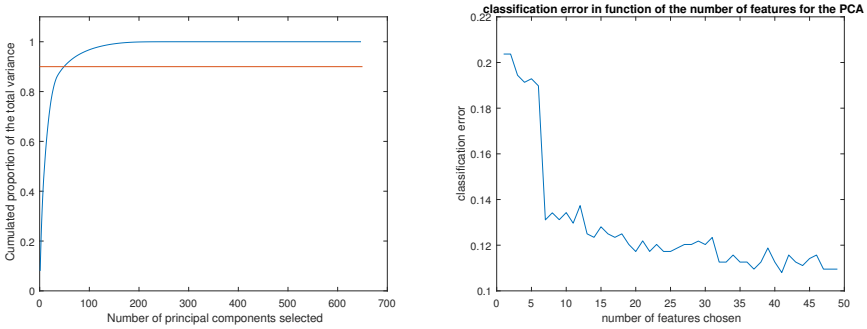


Figure 5: Left : Naive PCA of section 3.2.4 : Cumulated proportion of the total variance in function of the numbers of PCs. The threshold is equal to 0.9 leading to a selection of the 49 first PCs. Right : Evolution of classification error with a linear classifier according to the number of PCs. Cross validation of section 3.3.1

3.3 Cross-validation

3.3.1 Cross-validation for hyperparameter comparison

Cross-validation combined with PCA

We combined PCA and cross validation to evaluate the performance of PCA reduction in selecting the best features through cross validation. The maximal number of features we would look at chosen was equal to the naive result with a 0.9 cumulated variance proportion obtained in section 3.2.4.
Then for each fold of the cross-validation, the PCA was computed on the training set and the linear combination of features was then applied to both training and testing set. By this mean, the test set wasn't involved in the choice of the PCs, preventing from any bias. There, N_{PCA} was given a fixed value. Repeating this process for each value of N_{PCA} until a maximum value, we could then plot the successive mean values and look for an elbow point the choose the optimal N_{PCA} value (when the error reduction stops being significant enough to justify the selection of another feature). This is showed on figure 5 *right*.

The computational time is relatively long for a simple cross-validation, as the PCA has to be applied on each training set for all 2400 features. The results obtained can lead to low classification errors (around 0.12 with the linear classifier and 20 PCs), are stable over repetition and show a huge decrease in the classification error in the range [5, 10] PCs kept. A "steady state" in the error is roughly obtained in the range [15 - 40] PCs chosen. Selecting 20 PCs seems to lead to a good compromise between dimensionality size and error reduction. The corresponding classification error is then 0.12 with a linear classifier.

See code worksheet *Project02_PCA_inside_CV.m.*

3.3.2 Nested cross-validation for performance estimation

Nested cross-validation where the classifier type varies as a hyperparameter *on top* of the number of features

We used a nested cross-validation to determine the optimal number of features ranked by Fisher combined with the classifier type , the best "couple" classifier type/number of features was found to be N=5 features with the classifier type linear, with a minimal Testing error of 0,08. However this "couple" remains unstable over repetitions, we also had N=5 features when the quadratic classifier was selected (see Figure 6).

This is a very interesting result, as it seems that combining the best classifier type - thought to be linear (see section 3.1.2) - and the best number of features N=5 (see the result from section 3.2.3) - obviously gives the best combination.
However, this should be treated cautiously because of the instability of the result, and also because the maximal number of features to select was fixed to 5, meaning that we could actually have more than 5 features selected by the algorithm. Thus, hyperparameters influence each other and do not work independently, making it crucial to test them together to find the model that lowers the

Repetitions	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
Number of features (Fisher score)	5	5	5	5	5	5	5	5	5	5
Classifier type	quad	linear	linear	quad	linear	linear	linear	linear	linear	quad
Testing Error	0,09	0,12	0,17	0,09	0,11	0,15	0,09	0,11	0,08	0,16

Figure 6: Nested cross-validation with $k_{in} = 5$ and $k_{out} = 10$, showing the best couples classifier/number of features. These couples are not stable over repetitions.

classification error.

Statistical significance: mean Testing Error = 0,11 with a standard deviation of 0,05. As data is normally distributed (Kolmogorov-Smirnov test 2.3.3), T-test rejects the hypothesis of a distribution with mean 50% at the 5% significance level with a p-value equal to $6,9 \times 10^{-18}$.

See code worksheet *Project02_ Nested_ CV_ Fisher.m.*

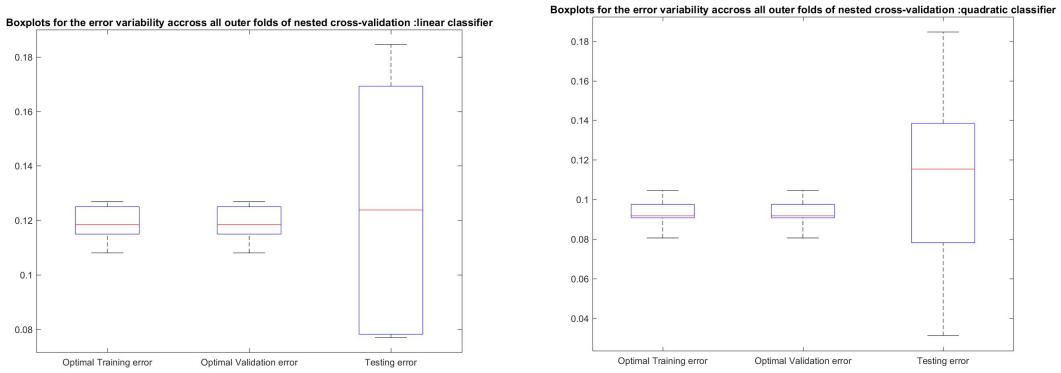
Nested cross-validation with PCA and Fisher method

This time nested cross-validation is applied with a feature selection based on PCA and Fisher method. We choose 10 outer folds and 5 inner folds, and we compared different types of classifiers. The hyperparameters are thus the number of PCs N_{PCA} and the number of features kept from Fisher method N_{Fisher}. We set an upper bound $N_{PCA,max} = 20$, in order to reduce the computational expense.

The principle is the following one: the PCA enables to reduce the dimensionality, and then the best of the PCs are kept using the Fisher method. For each inner cross-validation loop, a number of PCs is kept and the couple (N_{PCA}, N_{Fisher}) deriving from the lowest validation error is saved. This couple of values is then kept and the model is trained on the outer training subset and tested on the corresponding testing subset. A first result is that the best classifiers in this simulation are the linear and quadratic classifiers, leading to a mean classification error of 0.125 and 0.11 respectively.

Secondly, the table of figure 7 shows that the number of PCs kept in the process are roughly the same. After Fisher method, the linear classifier keeps a few less PCs (on average 14.5) but with a broader distribution (standard deviation of 2.9).

Last but not least, the figure 7 shows that, even if the test error of the quadratic classifier is the best one, the higher difference between training and testing error reveals a probable over-fitting of this classifier. This confirms the choice we have made in section 3.1.2 of keeping the linear classifier for our purpose.



	PCs post PCA		PCs post Fisher method	
	mean	std	mean	std
linear classifier	17,7	2,4	14,5	2,9
quadratic	18,6	1,2	17,2	2

Figure 7: Comparison for the linear and quadratic classifiers in the nested cross-validation of section 3.3.2 . Errors for the linear (upper left) and quadratic (upper right) classifiers ; comparison N_{PC} and N_{Fisher} over the 10 outer folds (lower figure)

Statistical significance: mean Testing Error = 0,16 with a standard deviation of 0,03. As data is normally distributed (Kolmogorov-Smirnov test 2.3.3), T-test rejects the hypothesis of a distribution with mean 50% at the 5% significance level with a p-value equal to $6,7 \times 10^{-11}$.

See code worksheet *Project02_Nested_CV_PCA_Fisher_2.m.*

Nested cross-validation with PCA and FFS

A second nested cross validation close to the previous one is run (k_{in} = 10, k_{out} = 20, $N_{PCA,max}$= 25). Here however, after PCA reduction, the feature selection is based on feature forward selection (wrapper method). Moreover, the classifier chosen is linear following the previous choices. N_{PCA} is the only hyperparameter. We indeed decided not to add the number of features kept after FFS as a hyperparameter but to choose it in function of a threshold in each inner training subset and outer training subset. As the corresponding *MATLAB* function wasn't available to our purpose, we coded by ourselves the feature selection. The threshold chosen was to stop the addition of new features if the classification error didn't get reduced more than 5.10^{-3} with one more feature. This threshold could have been encoded as another hyperparameter (varying and chosen within nested cross-validation; but we didn't do it due to computational time).

As shown on figure 8 The mean test error is 0,14. The corresponding number of features kept in the training outer fold is on average 3,9. Interestingly, the validation errors and test errors are very close, what might be explained by the fact that a new feature selection is realized in the outer fold. The error isn't as good as with Fisher method, but the features used are less numerous. Changing the threshold could lead us to different results.

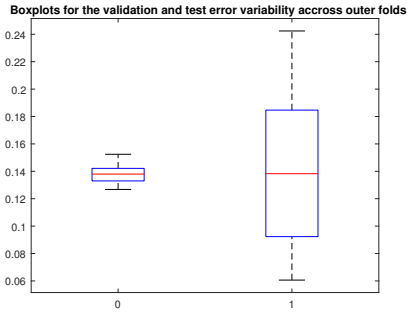


Figure 8: Validation errors (left) and test errors (right) for each fold of the nested cross validation with PCA and feature forward selection

Statistical significance: mean Testing Error = 0,14 with a standard deviation of 0,05. As data is normally distributed (Kolmogorov-Smirnov test 2.3.3), T-test rejects the hypothesis of a distribution with mean 50% at the 5% significance level with a p-value equal to $2,47 \times 10^{-17}$.

See code worksheet *Project02_FFeature_Selection_PCA.m.*

4 Discussion

This project introduced us to different methods for optimizing classification of data in the context of supervised learning. Different feature selection methods (feature forward selection, fisher score), different type of classifiers as well as dimension reduction algorithm such as Principal Component Analysis were analyzed throughout the project.

We have learned that all these hyperparameters are not independent. Making one hyperparameter vary on top of the others might significantly change the model. Therefore, how could we possibly design the final "best" classifier ?

One would think that combining **all** these different techniques on for example, a nested cross-validation with the variation of an hyperparameter on top of the others, would provide us the "best" model. However, we must underline that the computational cost of all these techniques is high, and the combination of them dramatically increases this cost. Furthermore, we have noticed

that combining these techniques requires a lot of data, and our data might not be sufficient to allow the use of all the techniques at the same time. Indeed, each cross-validation partitions the data into smaller subsets as we can see it on the summarizing scheme , and testing on very small subsets decreases the quality of our model. Being able to create the "best" model suggests that you have access to a large amount of data that you are able to split into smaller subsets without losing much information.

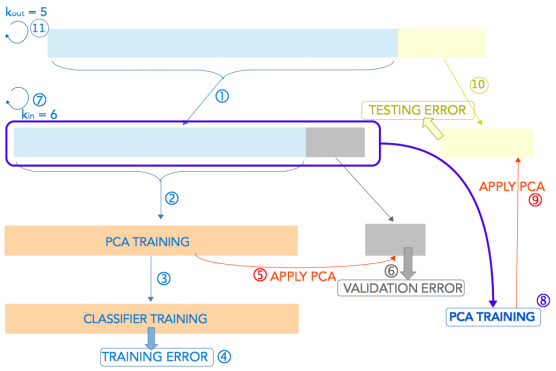


Figure 9: Slicing of our data into subsets for different loops of Nested CV, using PCA, Fisher selection on the reduced space and classifier selection.

For all these reasons, we have to make choices in order to select a "best model". Which hyperparameters to use? In this reasoning, we prefer not to draw much upon comparisons of the results we got from the different simulations as they didn't always involved the same numbers of inner/outer training and test sets. In order to properly design the best classifier, many different combinations of hyperparameters should be tested on the same draft of nested cross validations. As explained above (see section 3.1.2) , we assumed after several comparisons of the classifiers, that the best of them to our purpose was the linear classifier. Until now, almost all our nested cross-validations were performed under this assumption, which permitted us to keep the classification error low and an efficient classifier on an unseen data. Thinking about feature selection methods, we know that filter methods are much faster compared to wrapper methods as they do not involve to train the models. Wrapper methods are in fact computationally very expensive due to the repeated learning steps and cross-validation. However, although their computational cost is lower, filter methods are not as efficient as wrapper methods. This is why we thought about combining the Fisher score (filter method) with a PCA to increase the performance of this method while keeping the computation fast enough. The results from the PCA showed us that reducing the dimensionality of the space can be highly relevant. This can be followed by a FFS on the reduced dimensional space. This leads to a better solution than using FFS alone, as it balances algorithm performance and computational cost. Comparing the combination PCA/Fisher and PCA/FFS, we have similar mean Testing errors but a smaller standard deviation for PCA/Fisher which accounts for a better performance.

Finally, to assess our "best" model, we would perform a nested Cross-Validation with a linear classifier, in which the dimensionality of the features space would be reduced in the inner loop by a PCA, and followed by a Fisher score ranking to select the relevant features.

The results of our statistical analysis on the test error values across *outer* folds helped us to evaluate the performance of each nested *CV*. However, this analysis was performed on small subsets of data which makes the statistical analysis less viable. Moreover, one must underline that checking normality in small data sets might be hard: formal testing has low power so violations may not be detected well. With no way to reliably verify the normality assumption in a small sample, we should stick to non-parametric methods, which are thought to have a higher power than the t-test. [Bla80]

References

[Bla80] James J. Blair, R. Clifford; Higgins. A comparison of the power of wilcoxon's rank-sum statistic to that of student's t statistic under various nonnormal distributions. *Journal of Educational Statistics, 309–335*, 1980.