Mini-Project I

EE-516

Data Analysis and Model Classification

Cécile Crapart, Gabriele Gambardella, Eleonora Martinelli

Lausanne, 12th November 2018

Contents

1 Introduction

Supervised learning is a machine learning method in which the samples to be classified are provided along with the corresponding target class value (also called *label*). Here, an Event Related Potentials (ERPs) dataset is investigated, where each sample corresponds to the electroencephalogram (EEG) signal for an observed cursor movement and the two classes in which samples are divided are *correct* (70% of the samples) and *erroneous* (30% of the samples) movements of the cursor. More precisely, participants had to watch a cursor moving either toward the same direction as indicated by the cue or on the opposite direction, eliciting an ERP. On the basis of the provided dataset, this work aims at building a classification model able to optimally assign a class to each sample and which could be used on any unseen ERPs dataset with satisfying accuracy.

2 Methods

Generally speaking, a model is a function of parameters, such as means and covariance matrices, and hyperparameters, such as classifier type and dimensionality. The parameters are defined by training the model onto a given training set once the hyperparameters have been fixed. The optimal set of hyperparameters as well as the performances of the method used are defined through the process of *model selection*. More precisely, the output hyperparameters set will be the one optimizing a defined *model selection rule* fixed at the beginning of the process. So, the feature selection method and the model selection rule have been fixed at the beginning of the process. Then, the model selection process has been focused on the optimization of two different hyperparameters: the **classifier type** and the **feature number** (or dimensionality). When the number of samples in the starting dataset is sufficiently high, the dataset is split into training set, validation set and test set to perform the model selection. The ERP dataset is a 2D matrix made of 597 samples and 2048 continuous-value features (signal amplitude for sixteen EEG channels at different time points): each sample is associated with a label, where "1" corresponds to "erroneous movement" and "0" to "correct movement". The limited dimension of the ERPs dataset does not allow the above-described pathway; thus we have performed (1) Nested Cross-Validation (NCV) to evaluate the performances of the model selection method used, (2) Simple Cross-Validation (CV) for hyperparameters optimization and (3) lastly, the final model has been trained on the whole ERPs dataset based on the otipmal hyperparameters set found in the CV.

2.1 Preliminary decisions

2.1.1 Features selection method

Training the classification models on the entire dataset may result in overfitting the model because the number of samples is rather small compared to the feature dimensionality (often referred as the curse of dimensionality). In case of over-fitting, the classifier would fit the training data too closely and fail to generalize. Moreover, the higher the dimensionality the higher the complexity of the model, which might result in unnecessary high computational time and computational cost. Those drawbacks can be avoided by reducing the number of features but this requires to decide which features and how many to remove. Indeed, removing too many feature would result in under-fitting (poor mapping of the data). To address these issues we decided to extract new features by Principal Component Analysis (PCA), rank them depending on their discriminability using Fisher score, and finally optimize the number of most disciminative Principal Components (PCs) within the inner loop of the NCV. We also decided to use only a subset of the orginal dataset and considered features from 1 to 2048 with a step of 10. The reasoning behind was that features here represent different timepoints in the experiment conducted, therefore it was likely that features which are one next to each other would carry similar information (i.e. their covariance was high).
We used PCA for feature extraction because this allows to keep all the information contained in the original dataset and use it to engineer new features. Indeed, all feature contribute to each principal component (linear combination of all original feature weighted by a certain *loading*). The orthogonal

transformation only converts the set of possibly correlated variables into a set of principal components which are linearly uncorrelated. Although the PCs are difficult to relate to the original data, there was no clear need to interpret the features used and to evaluate what PCs physically correspond to in this case. PCA seeks to maximize the variance of each component so the first PC will explain most of the variance in the data and the following ones will explain the remaining variance. To overcome the problem of having different magnitude for different features, the PCA was computed following data normalization (see in the Pipeline). However, the variance explained by the first PCs could be related to noise and artifacts of the EEG recordings. Since we aim at keeping the significant features and excluding the features that carry the less amount of information regarding the prediction problem, we decided to implement *Fisher ranking* of the PCs. This aimed at only retaining the PCs with high discriminability. Feature selection could also be performed using a sequential algorithm (forward or backward feature selection). However, these are computationally more complex and very time consuming (only 10 features evaluated in 10 minutes when we applied FFS). We expect to obtain similar performances by combining PCA and Fisher ranking (the whole code takes less than 5 minutes to run).

2.1.2 Model selection rule

Let's assume that the dataset has been divided into train, validation and test set, and that different models (based on different hyperparameters subsets) have been trained on the training set. Then the hyperparameters are optimized according to the model selection rule. In this work two model selection rules have been investigated: the minimization of the classification error on the validation set and the minimization of the class error on the validation set, where classification and class error are defined as follows:

(1) $classification\_error = \frac{\#misclassifiedSamples}{\#totalSamples} = 1 - classification\_accuracy$

(2) $class\_error = \frac{1}{2} * \frac{\#misclassifiedSamplesClassA}{\#totalSamplesClassA} + \frac{1}{2} * \frac{\#misclassifiedSamplesClassB}{\#totalSamplesClassB}$

As it is possible to grasp from (1), if the classes in the starting dataset are not equally represented, the classification error will give a higher weight to the misclassified data of the most numerous class. On the opposite, the class error will give the same weight to both classes. In the ERPs dataset the two classes are not equally represented, since 70% of samples belong to class A while only 30% of samples belong to class B. However it is not possible to predict that this imbalance between classes would be kept in any new unseen dataset. Therefore, the model selection rule chosen has been the minimization of the class error obtained onto the validation set. While implementing the method on MATLAB, the prior probabilities for each class had also to be specified, choosing between *empirical* and *uniform*. The prior distribution represents prior beliefs about the distribution of the parameters. The more informative the prior, the more data are required to "change" these beliefs. By using *empirical*, the class prior probabilities are the class relative frequencies whereas with *uniform*, all class prior probabilities are the same (equal to 1/K, where K is the number of classes). To improve the generalization abilities on any unseen dataset with unknown classes division, the prior was set to *uniform*, which was also coherent with the choice of minimizing the class error.

2.2 Classifier types

We implemented our model using the MATLAB function **fitcdiscr** with different classifier types: linear, diagonal linear, quadratic and diagonal quadratic. Linear classifiers assume an equal covariance matrix for all the classes as opposed to quadratic classifiers. Diagonal classifiers further assume that features are uncorrelated and their covariance matrix are diagonal matrix. Discriminant classifiers are set to minimize misclassifications and assume that the conditional probability density functions are normally distributed. Using the Kolmogorow-Smirnow-Test, we could confirm that the distribution of samples within each class was normal for all the features. These classifiers are based on the concept of finding the best linear or quadratic combination of features to distinguish between the classes. We

had to use pseudo inverse since the dimensionality of each data vector exceeded the number of samples in each class, the covariance estimates do not have full rank, and could not be inverted.

2.3 Pipeline

Before starting any operation on the dataset, we decided to reduce the dimensionality by selecting one feature every ten in the starting dataset. Then, Nested Cross-Validation, Simple Cross-Validation and final model building were performed. To partition the dataset in folds as necessary for Nested Cross-Validation and Simple Cross-Validation, the MATLAB built-in **cvpartition** function was used, based on the samples labels.

2.3.1 Nested Cross-Validation

NCV was first performed, in order to evaluate the performances of the model and the stability of the model selection rule presented previously. The goal of NCV is to obtain an unbiased estimation of the performances of the model, because the optimization of the hyperparameters is done on the inner folds, while the performance estimation is done on the outer folds. We decided to divide the dataset in ten outer folds, and to have five inner folds. The division was carried on through the **cvpartition** function: initially the function was applied to the whole dataset in order to divide it into outer folds, then, for each outer fold, the function was used to define inner folds.

Inner folds. For each fold, training and validation sets were identified. The train set was first normalized using the MATLAB built-in function **zscore**, and then PCA was performed on it. After that, the features resulting were ranked by their discriminative power following the *Fisher* score, with the function **rankfeat**. The validation set was also "rotated" in the new feature space found by the PCA. To do so, normalization of the validation set was required, but the function **zscore** could not be used: in fact, in a real situation, the test set would be totally unseen, thus it would be impossible to normalize it using its own mean and standard deviation which would be unknown. For this reason, we subtracted to each element of the validation set the mean of the train set, and divided this for the standard deviation of the train set. The normalized validation set was then rotated in the new space by using the coefficients found by the PCA (centered value*coeff). For each of the four classifier types mentioned above, the model was trained on the "transformed" training dataset. This was done by taking into account an increasing number of features, ranging from 1 to 50. The class error was computed for all classifier types and number of features, and averaged among inner folds. The hyperparameters chosen were the one minimizing the class error on the validation set.

Outer folds. After having optimized the hyperparameters on the inner folds, we computed the final performance of the model on each outer fold. Again, the outer training test was normalized and PCA was performed on it. The outer test set was normalized as well, using the same procedure as before. Then, features were ranked according to their *Fisher* score and the resulting model was tested on the rotated test set. At the end of the outer loop, we obtained a $1*K_{out}$ vector, where K_{out} is the number of outer loop folds, of which we computed the mean and standard deviation. Finally, we performed a two-sample t-test using **ttest2** function to test the hypothesis that the test error values across outer folds came from a distribution with mean 50%, in order to check that our error was significantly different from the random error, which indeed was 50%.

2.3.2 Cross-Validation for hyperparameters optimization

CV was performed after NCV. Since the number of folds decided was ten, the dataset was split in ten subsets through **cvpartition**, nine of which were considered as training set, and the last one as test set. For each fold, normalization of the train/test sets, PCA and features ranking were done in the same way as described for NCV. Given the random nature of the partition and the small number of samples, the results from CV were different each we run the code. To fix this issue, the whole process of CV was repeated ten times and averaged the optimal number of features over the ten runs. The optimal classifier type was the one that appeared more often.

2.3.3 Final Model building

After having found the optimal hyperparameters (the number of features selected and the classifier type which gave us the lowest class error) we proceeded by building the final model. This has been done in a similar fashion as before. The original dataset was reduced; then, PCA was performed and features were ranked based on their Fischer score (after PCA). Finally, the model was trained on the whole dataset and with the selected features, and was tested on the provided test set, which was previously normalized as already the described in paragraph 2.3.1 and rotated in the new space found by the PCA. Once the final model was built, the predicted labels were extracted by using the model on the test set, and then uploaded on **Kaggle** for the competition.

3 Results

According to NCV, the expected performances of the method used are shown in Figure 1. The expected performances of the model selection method resulted in a class error of 0.3244 ± 0.0813, which is significantly lower than chance level ($p = 7.64e-5$ for the t-test).

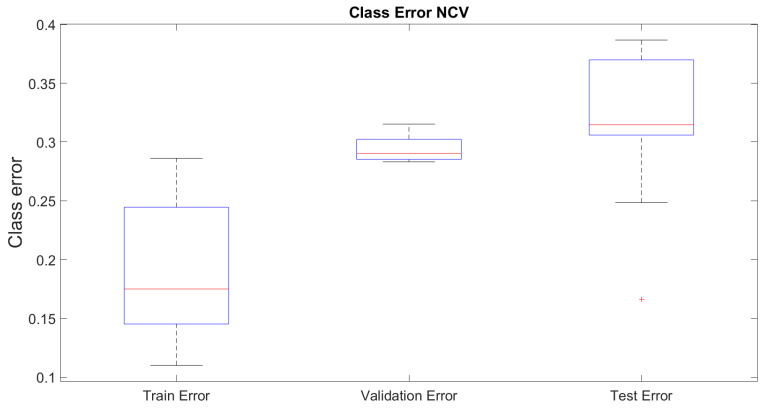

Figure 1: Boxplot of the train, validation and test error evaluated during Nested Cross-Validation. The median, the 25^{th} and 75^{th} percentiles, and the most extreme data points not considered outliers (whiskers) are represented. The train and validation class error refer to the model selected for testing on the outer fold. The test class error was significantly lower than chance level ($p = 7.64e-5$)

The optimal number of features selected to build the model was varying across outer folds as shown in Fig. 2, with median 21, mean 21.6 and std 13.201. This means that the model is not stable. The optimal classifier type chosen was also varying, with Diaglinear being chosen 4 times, Linear once, Diagquadratic twice and Quadratic 3 times.

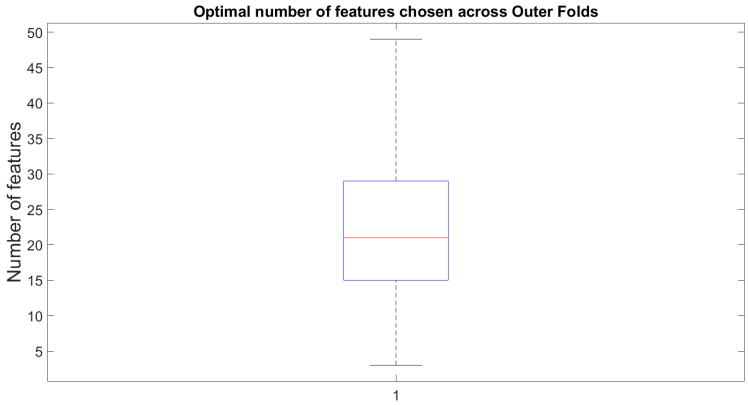

Figure 2: Boxplot of the optimal number of features selected across the 10 outer folds of the NCV.

Every time the Simple CV for hyperparameters optimization was run, a different set of best hyperparameters was obtained. It was therefore decided to repeat the CV multiple times (10) in order to fix the classifier type as the most frequent classifier type obtained. At the end of this process, we found the most frequent classifier type chose as the best was the Diaglinear (obtained 8 times out of 10); however, the corresponding optimal number of features was always very high (around 40), with a high chance of overfitting. For this reason, we decided to choose the classifier type leading to the lowest validation error among the ten repetitions done in CV. This lead us to choose a Linear classifier type, with a corresponding number of features of 17, which is coherent with the median optimal number of feature obtained from NCV (median of 21). Moreover, as we can see from Figure 3, the model starts overfitting with a higher number of PCs. Table 1 represents the different sets of hyperparameters obtained for Simple CV and their respective validation and training errors averaged across 10 CV folds. As we can see the Linear is the one with the lowest validation error.

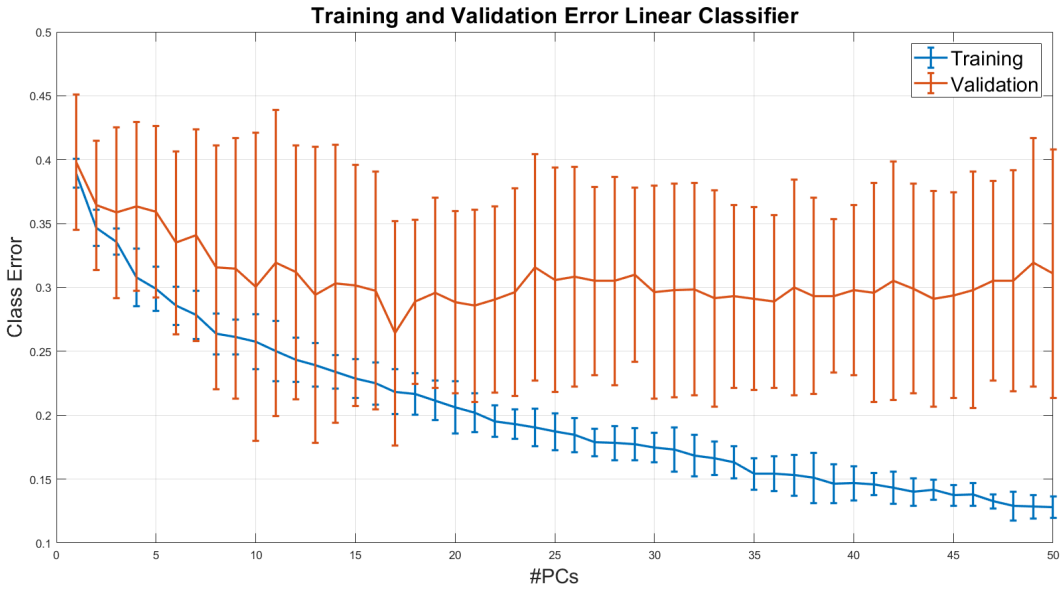

Figure 3: Training and Validation errors of the Linear classifier type, obtained in Simple Cross-Validation (10 folds), as a function of number of PCs considered. Mean class error and standard deviation are plotted.

Table 1: Results from Simple Cross Validation. It presents the optimal number of features, mean (± standard deviation) training and validation error across the 10 fold of the CV (iteration at which the the minimal class error was obtained and linear classifier was selected).

Classifier type	**Optimal # of features**	**Mean training error**	**Mean validation error**
Diag linear	17	0.2187 ± 0.0180	0.2654 ± 0.0870
Linear	17	0.2184 ± 0.0174	0.2643 ± 0.0878
Diag quadratic	40	0.1402 ± 0.0169	0.2891 ± 0.0798
Quadratic	14	0.1853 ± 0.0134	0.2777 ± 0.0986

In the designed model selection method, considering the small number of samples available compared to the high number of features, features reduction was a critical step to prevent overfitting of the final model. To do that, PCA was used in order to reduce the dimensionality of the data while preserving the variation present in the dataset as much as possible. However, as previously mentioned, the variance in EEG data could be mainly due to noise. That is why Fisher ranking was performed afterwards. We reasoned that if discriminability in the data was perfectly related with variance, then PCA with Fisher and PCA alone should result in PCs order. However, this was not the case for this dataset: after Fisher ranking the PCs were reorganized as follows (1 3 13 71 8 91 73 49 47 62 107 2 35 5 45 37 7).

4 Discussion

In summary, we have performed feature extraction and selection (PCA on a subset of feature followed by Fisher-score ranking) and we have optimized two hyperparameters (classifier type and number of feature selected) using Simple Cross-Validation to build the final model. This resulted in selecting the linear classifier with 17 features. According to Nested Cross Validation, we expect this model selection method to perform with a class error of 0.3244 ± 0.0813. Obtaining an accuracy of 67.56% is relatively satisfying using data from EEG recordings.

Many alternatives to the described pipeline for the dataset classification exist. EEG recordings can be contaminated with muscular or ocular artifacts such as eye movement and eye blinks. Moreover, EEG data derived from multi-channels (here sixteen) are carrying noise captured from adjacent channels, especially if channels are close to each other. By plotting random samples, we observed that the data were very noisy. Moreover, we have no information about how data were processed before obtaining the dataset. We hypothesized that the application of spatial filters that maximize the variance of EEG signals of one class while minimizing the variance from the other class of the data as preprocessing step could help reducing the noise captured from adjacent channels in the multi-channels EEG recordings. This, however, would not solve the issue that EEG recordings are contaminated by many artifacts including muscular or ocular artifacts.
One more advanced way to reduce EEG artefatcs would be to use Independent Component Analysis (ICA). This would better account for the fact that EEG is a multivariate signal of independent cerebral and non-cerebral sources. It would allow to discard the Independent components of artefactual sources (the automatic extraction of relevant ICs is challenging and might require supervision) and combining the remaining ICs, as we intended to do with Fisher ranking.
Another way to address the issue of dimensionality is to use regularized discriminant analysis to adjust the covariance matrix. The aim is to replace the within-group sample covariance by a weighted average of the whole sample covariance. The regularization paramater (here 'Gamma' parameter in the **fitcdiscr** function) aims at increasing larger eigenvalues of the sample covariance matrix while decreasing smaller ones (the eigenvectors corresponding to the smaller eigenvalues will tend to be very sensitive to the exact choice of training data). It can be optimized by Cross Validation. We tried to implement that kind of model (classifier type was fixed to LDA in that case and features used were not transformed). According to Nested Cross Validation, we can expect this kind of model (which optimize Gamma) to perform with a mean class error of 0.2877 ± 0.0618 on the outer folds. The best entry with regularized LDA scored 0.80198 on Kaggle.
Another aspect which could improve the dimensionality reduction of the dataset would be having a better knowledge about its composition. As a results, some features could be safely not considered without losing any information simply because of the experimentation set-up used.
Finally, we could also have tried different families of models: a nearest neighbor classifier, a support vector machine (SVM), a decision tree or even a neural network.