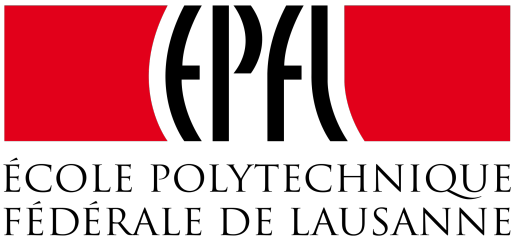
Ecole Polytechnique fédérale de Lausanne

Mini-project 3:
EE516: Data Analysis and Model Classification

Final Report

Constantin Krempp, Alexandre Delaux and Anastasia Bouzdine

December 4^{th} 2017



1 Introduction

Regression analysis is an important statistical method for the analysis of medical data, and is widely used in fields like neuroprosthetics. It enables the identification and characterization of relationships among multiple factors. The dataset used in this project originates from an invasive brain machine interface experiment where a monkey is moving a pole with its arm, while a number of electrodes implanted in the monkey's brain are used to obtain neural firing rates corresponding to the arm movements. The dataset *Data.mat* contains the Time samples × Features *(48 neurons × 20 50-ms bins)* of neuron spike rates, *PosX* and *PosY* representing the Cartesian coordinates X and respectively Y of monkey's wrist.

In this project, we aim to regress the arm movement trajectory to the neuronal firing rate of a monkey from the past recordings of neural activities in its brain. Several methods will be introduced to perform the regression and a definitive model will be chosen based on its performance in rightly predicting the movements of the subject.

2 Methods

2.1 Regression Methods

Regression can be used in a supervised learning problem where y is a continuous variable whose values are predicted from measured input variables x, whereas in a classification problem, y is a discrete variable denoting the type of class we want to assign a sample x. Therefore, it attempts to define a mapping f : x → y such that the error between Y and f(X) is minimized.

In this project, we will assess the performance of regression models for predicting two different variables *PosX* and *PosY* from *Data*.

2.1.1 Univariate Linear Regression

Linear regression methods predict the value y_n given an input vector x_n through a linear function f(x_n,w) with respect to the computed parameters w.

$$y_n = f(x_n, w) = \sum_{i=0}^{P} w_i \phi_i(x_n) \tag{1}$$

where $\phi(x)$ is any function (even a non-linear one) of the input vector x. In the univariate linear case, there is a unique input variable x and $\phi_0(x) = 1$, $\phi_1(x) = x$.

Univariate Linear Regression was performed for both *PosX* and *PosY* using *MATLAB* function *regress*.

2.1.2 Polynomial Linear Regression

In Polynomial Linear Regression, the powers of the input are up to a certain degree P , instead of just using the input x_n and the intersect term. In this case, instead of fitting a line to the input/output pairs, we attempt to fit a polynomial of degree P to the input/output pairs. The function ϕ is therefore defined as $\phi_i(x_n) = x_n^i$. The linear regression model in this case will have the form:

$$y_n = f(x_n, w) = w_0 + w_1 \times x_n^1 + w_2 \times x_n^2 + \ldots + w_P \times x_n^P \tag{2}$$

The same *MATLAB* function *regress* was used.

2.1.3 Reducing the input space dimension

When the input space has a high dimension, several risks can occur : need for a large amount of training data, overfitting, high computational cost, etc. To avoid these problems, reducing the dimensionality by projecting the original data onto a lower dimensional space seems to be a good solution. To do so, Principal Component Analysis (PCA) transforms the original variables to the linear combination of these variables which are uncorrelated, and provides a reduced dimensional space, where features are sorted by the decreasing amount of variance they explain.

To further reduce the number of features, we can observe the behavior of our regression model when this number of features gradually increase. We can then define a threshold to determine how

many features to select in the purpose of reducing the error by a given percentage p of the total reduction available (see equation 3). This percentage was set to 90% in practice.

$$N_{features}\ s.t.\ Err(N_{features}) = (Err(1) - Err(Max_{features})).(1-p) + Err(Max_{features}) \quad (3)$$

2.2 Regularized regression methods

In order to prevent overfitting in the case of few samples compared to the number of features, regularization methods can be employed. These techniques penalize large values of the weights in the loss function by restricting a multiple of the L_1 or the L_2 norm of w to 1. The result is a decrease of the complexity of the model. This method is particularly interesting when the sample size is small against the number of parameters to evaluate, therefore only 5% of our dataset was used for training to get the true benefits of this method.

2.2.1 LASSO regression method

In the LASSO method, large weights are penalized using a constraint of the following shape:

$$\min_\beta ||y - X\beta - \beta_0||_2 \quad s.t. \quad \lambda||\beta||_1 \leq 1 \quad (4)$$

This method leads to sparser models with fewer coefficients. In particular, many weights will become zero when λ increases. λ is a new hyperparameter that needs thus to be determined in the model. The lasso function proposed by *MATLAB* uses a cross validation to automatically select the λ giving the smallest cross-validation error.

2.2.2 Elastic nets method

A generalization of LASSO is also used to cope with some drawbacks related to LASSO method (in particular the fact that when groups of features are correlated, LASSO will only select one of the features since it optimizes sparsity). The function to optimize remains the same (regression) but the constraint is modified:

$$c(\beta) \leq 1 \quad with \quad c(\beta) = \alpha||\beta||_1 + \frac{1-\alpha}{2}||\beta||_2^2 \quad (5)$$

A second constraint is added on the L_2 norm and the two constraints are weighted by the factor $0 \leq \alpha \leq 1$. LASSO method corresponds to the case where $\alpha = 1$.

2.3 Selection of the final model

Each of the regression methods above can be eventually applied to our dataset and optimized by choosing the best set of their respective hyperparameters. At that point, they will be compared to determine the most efficient final classifier. For each of the methods, we applied a 20-fold cross-validation to extract mean training and test errors for $PosX$ and $PosY$.

A crucial question remains : which metric to use to assess the performance of our model ?

The Testing Error ($testErr$) is representative of how the classifier behaves on an unseen data set. But in parallel, the absolute value of the difference between the Training and the Testing error ($diffErr$) provides us insight on the stability of the classifier's reliability by quantifying the risk of overfitting. In order to be efficient, we want our classifier to show at the same time a low $diffErr$ and a low $testErr$. To that purpose, the following metric $Perf$ was implemented:

$$Perf(model\ M) = testErr(M) + a * |diffErr(M)| \quad (6)$$

The number a is a weight rectifier so that the influence of $diffErr$ on $Perf$ is balanced. a was set to 10 by ourselves. Typically, $diffErr$ represents a fraction of $testErr$: if this fraction is too important (at least 10%) we want our performance to penalize the model and by multiplying $diffErr$ by 10, it has the same order of magnitude as $testErr$ and can then weight heavily in $Perf$. When the fraction is lower, the order of magnitude between $testErr$ and 10*$diffErr$ is high and $testErr$ has the main weight in $Perf$.

Using cross validation, we train and test a certain number of models (defined by a set of hyperparameters) for $PosX$ and $PosY$ and compute the average value over the folds of the training and test error for each model. It leads to a value of $testErr$ and $diffErr$ for each model. We can then evaluate the performance of each model and select the one **minimizing** it to have the best according to this metric. The performance metric will also be used to compare the two main methods of regressors introduced.

3 Results

3.1 Regression Methods

We performed a data partition between a training set and a testing set. Normalization was applied independently on the training and testing set in order to reduce the risk of producing misleading results and missing statistically significant terms. Given the percentage of variation that we want to be captured in the abridged data set – 90% – we were able to select the number of Principal Components to be considered : N = 741 features were selected.

To learn a regressor, *MATLAB* function *regress* was used. The regressor was then trained on the data projected on the PCs, as it provided us an already reduced space where variables were classified according to their variance.

3.1.1 Univariate Linear Regression

The regressed vectors and the real vectors were plotted for both $PosX$ and $PosY$ using *MATLAB* function *regress* (see figure 1).

Univariate Linear Regression was evaluated for both $PosX$ and $PosY$ through *MATLAB* function *immse* on the training and the testing set. This function evaluates the performance of the regression by returning the mean-squared error between the y vector and the regressed on. Using this naive approach, we found a Training Error$_{,X} = 3,5 \times 10^{-4}$ and Testing Error$_{,X} = 4,2 \times 10^{-4}$ for $PosX$ and a Training Error$_{,Y} = 1,9 \times 10^{-4}$ and Testing Error$_{,Y} = 2,4 \times 10^{-4}$ for $PosY$. A higher error is obtained using the Testing set which corresponds to the "unseen" data. The variability between $PosX$ and $PosY$ can be due to inaccurate measurements or external factors. In general, a higher error characterizes the $PosX$ vector, and this is why we will mainly present the results using $PosX$ further in the report.

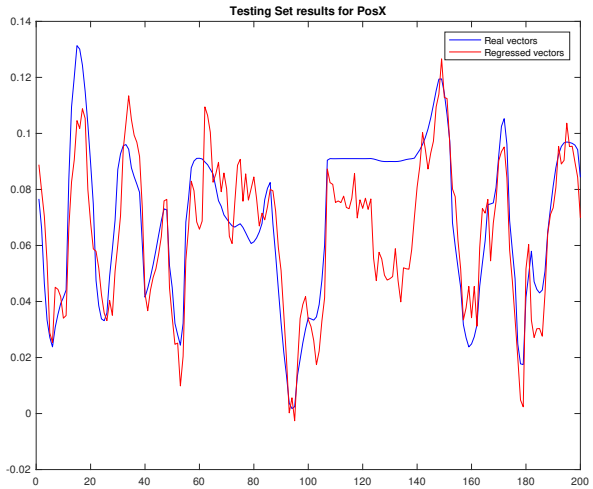


Figure 1: Visualization of the regressed vectors (red) superimposed with the real vectors (blue) $PosX$ and $PosY$ on the Testing Set with a linear regressor.

See code worksheet *Project03.m*.

3.1.2 Polynomial Linear Regression

After this first overview of our dataset, we wanted to investigate further the influence of the polynomial order of the regressor. To that end, feature reduction was necessary so we sorted our features with the PCA algorithm. Gradually, the number of features used to train the regressor was increased one by one until reaching a maximum fixed by ourselves. This allowed us to implement the criterion from section 2.1.3 to select a reasonable amount of features. The five first polynomial orders were successively explored and the overall process was introduced in a cross validation loop with 20 folds to work with consolidated numbers.

However, comparing the results obtained is not that easy: in fact, plotting according to the number of features used is not a good way to represent the performance of each model. What matters most is the number of parameters used by the regression model, which follows the formula: $N_{param} = D * N_{features} + 1$ where D is the polynomial order. This note was taken into account to plot Figure 2.

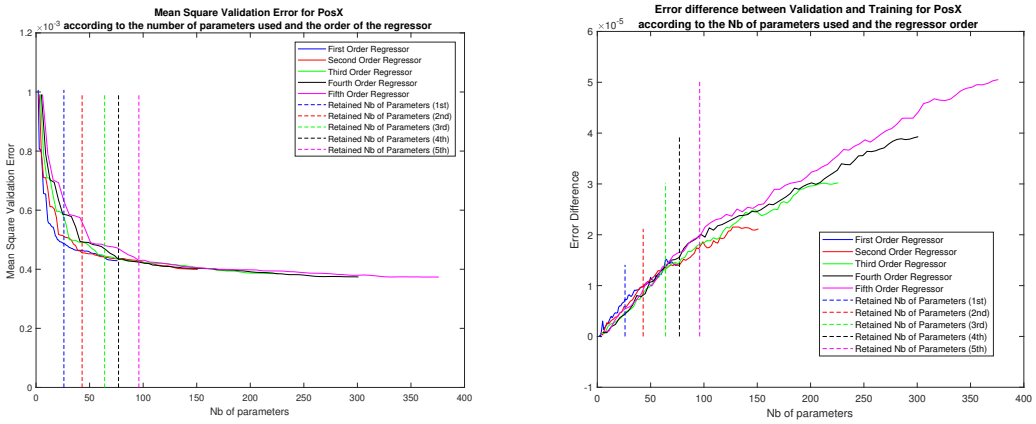


Figure 2: (left) Validation Error according to the number of parameters used by the regressor.(right) Difference between Validation and Training Errors according to the number of parameters used by the regressor.

As expected (Figure 2 left) the error diminishes while incorporating more and more features (equivalently parameters) to the model. However, the error gain is fast at the beginning and rapidly decays. This justifies the fact of selecting the number of features allowing to explain 90% of the maximal error gain, as it will drastically reduce the dimensionality. All polynomial orders seem to have an equivalent asymptotic behavior (high number of parameters) a very slow and regular decrease of the error. However this graph is much interesting as it proves that the first order regressor uses much less parameters to reach the lowest error values.

On the right of Figure 2 is plotted $diffErr$, where we can notice that every polynomial order leads to the same behavior: a quasi linear increase according to the number of parameters, with a low value for few parameters : Testing and Training errors are very close at that point as the low number of parameters prevents a precise training which would introduce differences between 'seen' and 'unseen' data for error evaluation.

Figure 2 also presents (dashed lines) the number of parameters retained by the implementation of section 2.1.3 criterion. This number differs regarding the Training or the Testing Error, $PosX$ or $PosY$, and obviously the polynomial degree of the regressor. This threshold increases with the polynomial order, on the one hand reducing the error but on the other hand increasing the error difference. To that regard, the performance metric will be particularly important in selecting the optimal $(N_{feat}; D)$ couple there. This is interesting to note that these variable thresholds are in fact much less variable in terms of features: each polynomial order roughly needs the same number of features. For example in the case of Figure 2 $PosX$, $N_{featopt} = 25$ for the first order, 21 for the second and the third order and 19 for the fourth and the fifth order.

Evaluation of performance : The best couple (N_feat, D) *is* $(21, 2)$ *here. The corresponding training and test errors are respectively* 3.3961×10^{-4} *and* 3.4706×10^{-4}*, leading to a performance* $Perf$ *equal to* 4.2162×10^{-4}.

See code worksheet *Project03_N_PCA_CV.m*.

3.2 Regularized regression

As regularized methods are particularly useful in the case of small datasets, we trained them on only 5% of our dataset and tested the performance on the 95% remaining samples. In this section, we take the maximum advantages of these methods by going further in the exploration of Lasso method and Elastic net.

3.2.1 Interest of regularization

We first apply a naive regression without PCA. As for the X position, the training mean square error is equal to 6.0×10^{-32} and the test error is equal to 0.03. The training error is unusually low compared to the test error (10^{30} times lower) and the test error is also two orders of magnitude higher than in the previous regressions made. This shows a huge overfitting realized on the testing set. Such an overfitting isn't surprising, as the number of samples is 643 while 961 parameters are used for the regression This result highlights the need of new "regularized" methods to prevent overfitting in case of small datasets.

3.2.2 Optimizing regression hyperparameters with Lasso and Elastic Nets methods

We apply LASSO method and its generalization, Elastic nets method, to our dataset. (λ, α) are the hyperparameters that need to be optimized using our metric. For each value of λ and α, a 20-fold cross validation is applied to extract mean training and test errors. The case of LASSO method corresponds to $\alpha = 1$.

We observe that the training and test errors decrease when α tends to zero and λ increases. We choose to focus on the intervals $\alpha \in [10^{-7}; 10^{-4}]$ and $\lambda \in [10^{-2}; 10^{2}]$. The Elastic nets method is actually more relevant in the case that we are studying in comparison to LASSO method, as shown in the results.

As expected and showed, the number of features kept by the algorithm decreases when λ increases. The corresponding sigmoidal curve is showed on the figure 3.

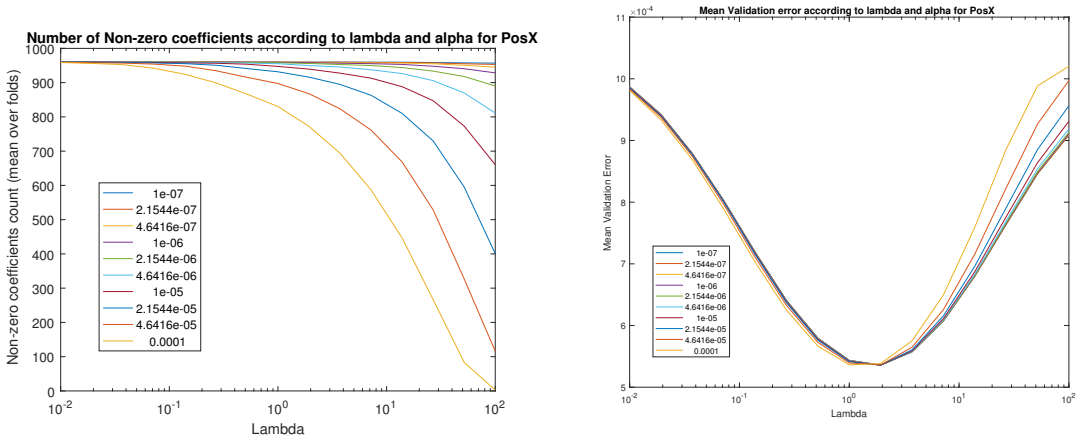


Figure 3: For PosX, comparison of the behavior of elastic nets method with $\alpha \in [10^{-7}; 10^{-4}]$ and $\lambda \in [10^{-2}; 10^{2}]$. The scale is semi-logarithmic. Left : number of non zero weights, right: validation error.

The test error according to λ is showed on the figure 3. The error first decreases until a certain λ_{min} as the model optimizes the weights. The increase after reaching the minimum could be explained by the fact that too few features are actually kept. There is not enough information to build a proper regression. A second observation is that the minimal test error and λ_{min} are very close for α in the range considered. This parameter has actually a low influence on the error.

Evaluation of performance : The best couple (λ, α) *is equal to* $(1.93, 2.2 \times 10^{-5})$. *The corresponding training and test errors are respectively* 4.17×10^{-4} *and* 4.11×10^{-4}, *leading to a performance* $Perf$ *equal to* 9.4×10^{-4}.

See code worksheet *Project03_ ElasticNets_ CV.m.*

3.3 Choice of the final model

As a large dataset was available in this experiment, we chose to take advantage of this data and use it entirely for our model. Regularized methods proved to be efficient in the case of small datasets; however in our case, both non regularized and regularized methods provided good results. As their implementation is mutually exclusive, we need to choose between them. The performance as computed with the metric introduced above indicates better results with the simple non regularized regression methods. This is why we selected the non regularized methods and set the number of features and the polynomial order as hyperparameters for our final model.

What we propose is a simple cross validation with the number of features and the polynomial order as hyperparameters. A Principal Component Analysis is performed on our data to reduce the dimensionality of the space and classify the features according to their variance. Two loops are introduced, one for the polynomial order D (until D=5) and the second one for the number of features which is gradually increased using the classified features from the PCA (until maximum value for $N_{feat} = 75$). The threshold we defined in section is then used to select the optimal number of features corresponding to each polynomial degree. Finally, the $Perf$ metric from section 2.3 allows us to choose the best polynomial order for our model.

To assess an unbiased statistical performances of the model, we also performed a nested cross-validation (10 outer folds and 10 inner folds) but the computation time was so long we cannot present its results in the present report (see code worksheet *Project03_ElasticNets_Nested_CV.m*).

Evaluation of the performance (simple cross validation): The best couple (N_feat, D) *is equal to* $(21, 2)$*. The corresponding training and test errors are respectively* 3.3961×10^{-4} *and* 3.4706×10^{-4}*, leading to a performance* $Perf$ *equal to* 4.2162×10^{-4}.

4 Discussion

The choice of the final model proved to be quite difficult because there were no obvious "best" decisions regarding the choice of the hyperparameters. Indeed, the different methods used in regression and regularized regression gave similar results regarding the Testing error and the performance of the model. However, we could definitely not mix all these different hyperparameters in our final model, and therefore we had to take decisions based on our intuition and the results obtained in previous sections.

For instance, although the results obtained with regularized methods (see section 3.2) showed to work well, the hyperparameters α and λ may be not appropriate in our case, where the dataset is big compared to the number of features.

Comparing the performance of a linear regressor and a polynomial order regressor when adding more parameters and features has shown to be a complex question. Indeed, we should take into account the complexity but also the generalization capabilities of these methods. Therefore, accuracy of classifiers should be assessed on a separate unseen test dataset, to see how the model performance holds for new, unseen inputs.

The metric introduced to evaluate the performance of our model is still questionable (see section 2.3). It allowed us to take into account the Testing error and the difference between Training and Testing error in a weighted manner. We chose a value of a equal to 10, what seemed to balance well the two components of the performance in our case. Increasing a would have more penalized the difference between training and validating error, and decreasing it would have however penalized a higher test error. For instance, $a = 1$ favors higher polynomial degrees for the regression, as overfitting has less weight into the metric. Thus, choosing a metric also requires to have a rough idea of the weight we want to give to those parameters, and the conclusion needs to be checked in order to make sure that the final model fits the initial expectations.