# Brain-Computer Interface (BCI) Trajectory Prediction

Andrea Bellotti
Politecnico di Milano
Piazzale Leonardo da Vinci, 32
Milan, Italy
andrea1.bellotti@mail.polimi.it

## ABSTRACT

People can learn to control their brain activity and use this control to move, for instance, a cursor on a screen. This project is a step in this direction. Specifically, the goal of the Kaggle competition [https://inclass.kaggle.com/c/brain-computer-interface-bci-trajectory-prediction] was to take readings of brain activity (voltages) and learn a mapping from these readings to velocity vectors for a computer cursor. Finding a mapping from brain activity to cursor velocity vectors is a non-trivial task. Currently, the state-of-the-art is to first perform spectral estimation on the voltage readings (this gives a sense of the frequencies of the signal), and then to perform regression to map these frequencies to the true velocity vectors.

Being a project for the Machine Learning course, I focused on improving the regression from spectrum to velocity vectors, avoiding to improve the spectrum estimation procedure, relying on the one provided by the organizers of the competition. I tried many different regression approaches but I obtained the best results with *Neural Networks* and *Gradient Boosted Regression Trees* ending up in 1$^{st}$ position of the leaderboard at the end of the competition (even if, to be honest, there has not been much competition since only 5 teams were registered and most of them made very few submissions). In particular I obtained the best absolute result with a 2-hidden-layers *Neural Network* which achieved a RMSE of 0.60218 against a baseline RMSE of 0.66689.

## 1. INTRODUCTION

Electroencephalography (EEG) is a non-invasive technique that allows to acquire electrophysiological activity of the brain. EEG, thanks to its simple and inexpensive setting, is widely used for neuroscience research and clinical applications. Electrocorticography (ECoG), or intracranial electroencephalography (iEEG), instead uses electrodes placed directly on the exposed surface of the brain to record electrical activity from the cerebral cortex. This means that it requires a craniotomy (a surgical incision into the skull) to implant the electrode grid, and therefore ECoG is an invasive procedure. A clear advantage of ECoG over EEG is that it possess both high spatial (mm scale) and temporal (ms scales) resolution. A further advantage of ECoG over scalp EEG is that intracranial recordings are not so susceptible to artifactual contamination from muscle movements and eye blinks, which regularly impair the quality of scalp EEG recordings.

Brain-Computer Interface (BCI) systems function by decoding the neural activity (EEG/ECoG) and translating the brain control states directly to output commands used to communicate with interfaced external devices. Recent developments suggest that the BCI users can develop control over components of their electroencephalographam by feedback-training provided by an adaptive BCI system [1], and this is very promising because in the future it could provide new communication and control options for people with severe motor disabilities.

Current approaches to EEG-based communication can be divided into two groups, those that use time-domain EEG components and those that use frequency-domain components [3]. Frequency-domain methods use spectral analysis and focus on specific frequencies at specific scalp locations. In the competition I adopted the second approach for two reasons: the first is that it was suggested by the organizers, and the second is that it is more suitable for applying the algorithms presented during the lectures of the Machine Learning course.

The provided dataset was composed of readings that were measured from the brain of a monkey while it watched a researcher move a cursor with a joystick from a START state to a GOAL state. While watching it attempted to (with its mind) to direct the cursor from the START state to the GOAL state (this was encouraged by giving the monkey a reward in previous trials every time the cursor reached the GOAL state. The setup is depicted in the figure below:

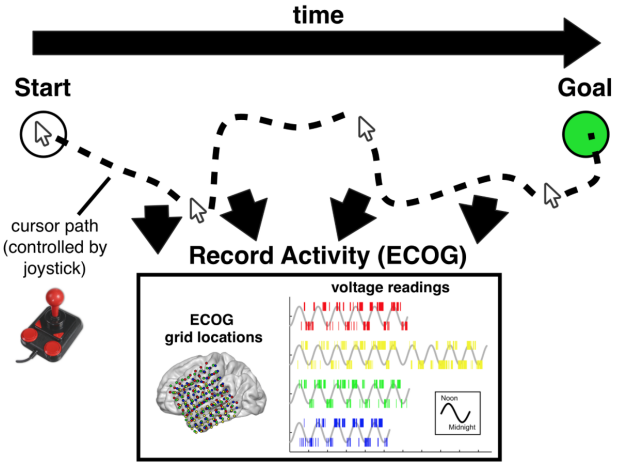


**Figure 1: The data acquisition setting**

## 2. PROBLEM FORMULATION

The goal of the project was to predict a multivariate output (x and y components of the velocity vector of a screen cursor) starting from an input composed of 32 simultaneous voltage readings collected in as many different areas of the brain surface (ECoG). To work in the frequency-domain a spectrum estimation was performed on the readings. This procedure transformed the 32 voltage signals into their counterparts in the frequency-domain, extracting the 5 most relevant bands while downsampling the input signals (2034.5 Hz) to the output signal (25 Hz) with a linear interpolation in order to have a correspondence. In addition to that, it was applied the z-score standardization to all the 160 (32*5) resultant features.

$$z = \frac{x + \mu}{\sigma} \tag{1}$$

The result of this process was a matrix $X_{31255\times160}$ that was used as input in all the machine learning algorithms that I tried. The problem therefore translated into finding a function $f$ able to approximate as well as possible the true function $\bar{f}$ that maps the input to the multivariate output.

$$\hat{Y} = f(X) \tag{2}$$

where $\hat{Y}$ is the multivariate output composed of the prediction of the x and y components of the velocity vector, and X is the input matrix obtained from the spectrum estimation procedure. The real goal of the competition was then to predict the multivariate output feeding the model with a specific provided test set.

## 3. METHODOLOGY

To solve the problem, initially I considered an hypothesis space $H$ composed uniquely of linear functions, focusing on linear regression. The first approach (baseline) I experimented was the basic Least-Squares $\hat{W}_{LS}$=(X'X)$^{-1}$X'$\hat{Y}$.

$$Loss_{LS}(w) = \frac{1}{2}\sum_{n=1}^{N}(y(x_n, w) - t_n)^2 \tag{3}$$

Then I tried three different regularization approaches: Ridge $\hat{W}_{RIDGE}$=($\lambda$I+X'X)$^{-1}$X'$\hat{Y}$, Lasso, and Elastic-Net, which linearly combines the L1 and L2 penalties of the Lasso and Ridge methods.

$$Penalty_{RIDGE}(w) = \frac{\lambda_1}{2}\left\|(w)\right\|_2^2 \tag{4}$$

$$Penalty_{LASSO}(w) = \frac{\lambda_2}{2}\left\|(w)\right\|_1 \tag{5}$$

$$Penalty_{ELASTICNET}(w) = \alpha\left\|(w)\right\|_2^2 + (1-\alpha)\left\|(w)\right\|_1 \tag{6}$$

where $\alpha = \frac{\lambda_2}{\lambda_1+\lambda_2}$ [5]

The latter two regularization techniques provided a significant improvement in the performance thanks to the main characteristic of finding more sparse solution because of the singularities on the vertexes, as Figure 2 shows.

Later I moved to non linear hypothesis spaces, experimenting with Neural Networks, SVM for regression, and ensemble methods such as Random Forest for regression and Gradient Boosted Regression Trees. The best absolute performance was achieved with a 2-hidden-layers Neural Network whose parameters (e.g., learning rate, number of neurons in the hidden layers, etc.) and activation functions were chosen with a 10 fold cross validation. The experiments with the two ensemble methods were an attempt to tune the bias-variance tradeoff: with Random Forest for regression it was employed the Bagging technique which consists in averaging the performance of different models trained with Bootstrap datasets (i.e., each model with a different dataset generated from the original dataset by randomly sampling with replacement) to obtain a reduction of the variance without increasing the bias. Due to the structure of the problem, the correlation between adjacent samples was very strong and as a consequence the random sampling of the Bootstrap produced poor performance. Instead, Gradient Boosted Regression Trees is based on the Boosting technique that sequentially training weak learners attempt to reduce the bias without increasing the variance, since weak learners have low variance. In practice, the performance achieved with this method is just a little bit worse than the one obtained with the Neural Network: 0.60442 vs 0.60218.

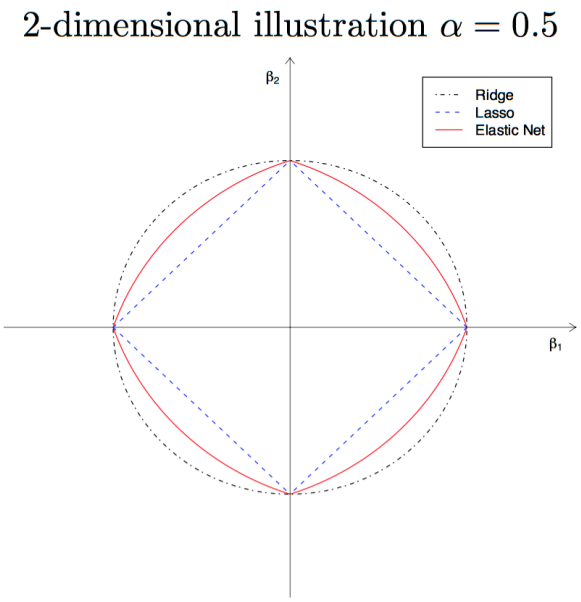


**Figure 2: Geometry of the regularizations**

## 4. EXPERIMENTS

As mentioned in the last section, to reach the 1$^{st}$ position of the leaderboard I experimented with many different algorithms, and apart from Least-Squares and Ridge, I took advantage of some open-source machine learning libraries written in python. The main two are *scikit-learn* and *pylearn2*, the last of which is built upon the well known *Theano* library.

To evaluate the performance of all the algorithms I used the RMSE (Root Mean Squared Error) metric because it was the one used in the Kaggle competition.

$$RMSE(y) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{7}$$

As previously outlined, the organizers of the competition provided one dataset for the training and one evaluation

dataset for computing the predictions to submit. The training set was initially composed of 31255 samples made of 32 features (voltages), but after the spectrum estimation (that included the down-sampling of the voltages to conform with the output measurement frequency) it was expanded to 160 features, corresponding to the 5 main frequency bands for each initial electrode. Each of these 160 features was also standardized with the z-score normalization. So, after the initial preprocessing, the training dataset was composed of 31255 samples, each with 160 features plus the two outputs (x and y components of the velocity vector). The evaluation dataset was pretty big compared to the training dataset, in fact it was composed of 28340 input values (also these were preprocessed with the spectrum estimation procedure to obtain the 160 standardized features).

Starting from the simplest algorithms I tried the closed form solutions of Least-Squares and Ridge regression, obtaining a baseline score of 0.66689. Then I tried to reduce overfitting applying the regularization techniques of Lasso and Elastic-Net. To tune the regularization parameter I performed a Grid Search with cross validation over the alpha parameter in the range $[1e^{-8}, 1e^{-1}]$ and obtained a best value $\alpha = 0.01887$ for Elastic-Net and $\alpha = 1e^{-2}$ for Lasso.

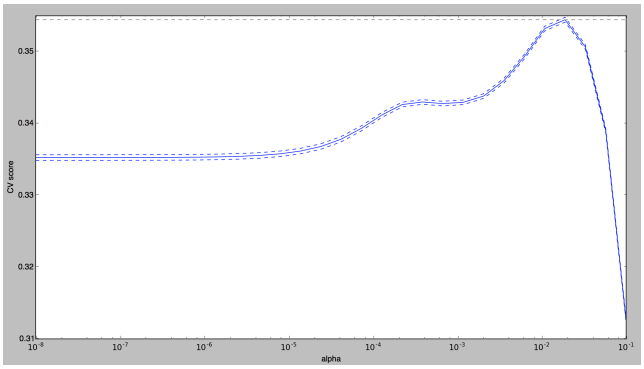


**Figure 3: Cross Validation Elastic-Net**

After these initial trials I switched to Neural Networks, which was the technique I wanted to experiment the most because it is widely employed in this kind of problems according to the literature [4] [2]. At the beginning I tried some manual configurations which resulted in very poor results so I built a Grid Search cross validation (3 folds) procedure to tune a feedforward 1-hidden-layer Neural Network with weight decay (regularization for avoiding overfitting).

| **Parameter** | **Values** |
|---|---|
| Number of Neurons | 5, 10, 25, 32, 50, 160 |
| Activation Function | sigmoid, tanh |
| Learning Rates | 0.01, 0.02, 0.03, 0.05 |
| Number of Epochs | 10, 15, 20, 30, 50, 100, 200 |

The best validation score (0.6486) was obtained with the following configuration:

| **Parameter** | **Values** |
|---|---|
| Number of Neurons | 5 |
| Activation Function | sigmoid |
| Learning Rates | 0.01 |
| Number of Epochs | 15 |

This model obtained a great improvement with the evaluation dataset because it scored 0.60258 where the second best score was the 0.61947 of the ElasticNet linear regression.

Then I tried to tune a more complex Neural Network with 2-hidden-layers, and as with the simpler one I performed a Grid Search over the parameters with cross validation, this time with 10 folds in order to try to obtain a more accurate validation error. With this setting I decided to try also different activation functions for the two layers, and in fact I discovered that the best configuration was to use in the first layer the sigmoid function, and in the second layer the tanh function. The final configuration found by means of the Grid Search with the same values for the parameters is the following:

| **Parameter** | **Values** |
|---|---|
| Number of Neurons (layer1) | 5 |
| Activation Function (layer1) | sigmoid |
| Number of Neurons (layer2) | 10 |
| Activation Function (layer1) | tanh |
| Learning Rates | 0.01 |
| Number of Epochs | 15 |

The validation error of this model was 0.6381 but in the evaluation dataset it scored 0.60218, which is the final best score of the competition.

The trickiest parameter to tune has been the dimension of the hidden layers, and the reasoning that led me to the values to try in the Grid Search was that since the inputs were 160 (32 electrodes $\times$ 5 frequency bands) it should have been a multiple of 5 or 32, because this would have grouped and connected similar bands and electrodes resulting in a more consistent learning.

After these two successful experiments I tried some other configurations, increasing the number of hidden layers and adopting different and more exotic activation functions, but no other significant result has been achieved, so I switched to a completely different method, Support Vector Machines (SVM) for regression. The experiment did not last long because the training time was enormous and therefore I couldn't perform cross validation to tune the hyperparameters (penalty C, and kernel). I tried a couple of configuration, including the default one of the *scikit-learn* library, just to make a rough comparison with the performance of the other methods, and in fact, without tuning the hyperparameters it performed poorly, just better tan the baseline. The default configuration sets the penalty C to 1.0 and the kernel to "rbf", which is the Radial Basis Function Kernel

$$K(x, x') = exp(-\frac{\|(x - x')\|_2^2}{2\sigma^2}) \tag{8}$$

At last I experimented a little bit with the two aforementioned ensemble approaches (Random Forest for regression and Gradient Boosted Regression Trees). In both cases the training time was lesser than the one of SVM but still not manageable (because of the competition and project time constraints) for a proper cross validation, so I reduced the search space and removed the cross validation procedure basing the manual tuning only on the training error. For the Random Forest approach I tried to tune just two hyperparameters: *max_features*, which is the number of features to consider when looking for the best split, and *n_estimators*, which is the number of trees in the forest. For the former I tried three different criteria: "sqrt", "log2", and None,

while for the latter I tried all the possible values between 15 and 175. As the following plot shows, the best configuration found is *max_features*=None, and *n_estimators* set to a value higher than 175 (but this most likely results in overfitting).

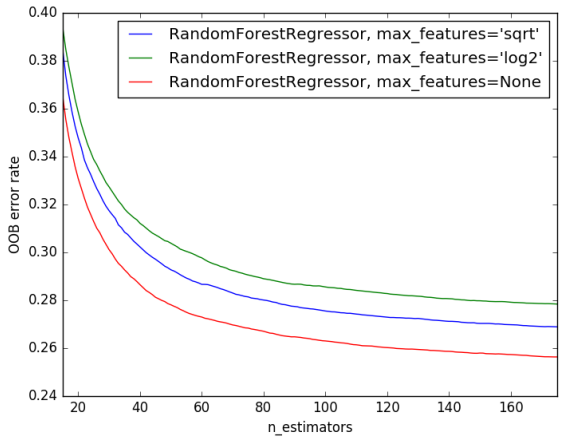


**Figure 4: Hyperparameters Tuning Random Forest**

This result is valid for both the output values to predict because the *sklearn.ensemble.RandomForestRegressor* algorithm is able to learn directly multivariate outputs.

A similar tuning process was applied to the Gradient Boosted Regression Trees method, with the only difference that it required to fit (and therefore to tune) a different model for each of the two outputs. This resulted in an even longer tuning and training time. At the beginning I tried to understand if reducing also the variance with subsampling (*subsample*<1.0), which is a bagging technique, could improve the original results achieved by GBRT that is intrinsically a boosting algorithm. Unfortunately, as Figure 5 shows, that was not the case, and the same held for the similar technique applied to the Random Forests, i.e., to reduce (via the *max_features* hyperparameter) the number of features to consider when looking for the best split in order to reduce the variance.

Once I figured out that the best setting had to keep the default values of the *subsample* and the *max_features* hyperparameters, I switched to the evaluation of the performance and the research of the best value for the *n_estimators* hyperparameter, which is the number of boosting stages to perform.

With the values of 812 for the $x$ model and 1120 (Figure 6) for the $y$ model I obtained a score of 0.60600, very close to the one achieved by the Neural Network. Subsequently I tried with the last submission to verify whether the result obtained in the first phase was correct, i.e., that subsampling (*subsample*<1.0) would have scored worse than the model with the default value *subsample*=1.0. Indeed, the result was wrong, in fact keeping all the other hyperparameters unchanged and setting *subsample*=0.3 the model obtained a score of 0.60442. This is most likely due to the fact that the result was wrong because it was based on the training error and not on a validation or test error. All the GBRT models were trained with the manually tuned hyperparameters *max_depth*=5 and *learning_rate*=0.01.

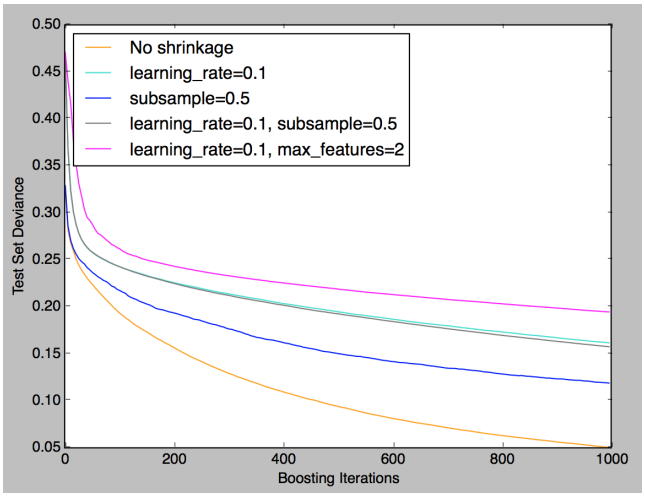


**Figure 5: Hyperparameters Tuning Gradient Boosted Regression Trees**

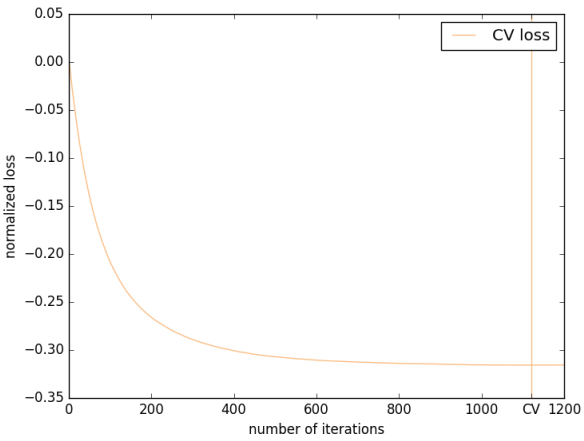


**Figure 6: Iterations Tuning Gradient Boosted Regression Trees**

The following table summarizes the results obtained with the various techniques:

| **Algorithm** | **Evaluation Score** |
|---|---|
| Neural Network (2 hidden layers) | 0.60218 |
| Neural Network (1 hidden layer) | 0.60258 |
| Gradient Boosted Regression Trees | 0.60442 |
| ElasticNet | 0.61947 |
| Lasso | 0.61963 |
| Random Forest for regression | 0.62511 |
| Bayesian Ridge | 0.63076 |
| SVM for regression | 0.64717 |
| Ridge | 0.66689 |
| Least-Squares (baseline) | 0.66689 |

## 5. CONCLUSIONS

This project shows that there are various machine learning methods which can achieve interesting performances in the multivariate regression task of predicting imagined trajectories. In particular two of them outperformed all the other methods and therefore are the one that should be investigated the most for improving even further the prediction performance. The Neural Network approach could be extended including convolution layers or used to learn the outputs based on both time-domain and frequency-domain features. The Gradient Boosted Regression Trees method, instead, has a much wider room of improvement, since the performance obtained in this experimental campaign was obtained with very little hyperparameters tuning due to the lack of computation power and time. Another possible approach which I didn't have time to try is the hybridization of the prediction of completely different methods, for instance with a weighted average as it is usually done in the Recommender Systems field, in order to take advantage of the strengths of each of them.

In general I'm satisfied about the outcome of the project because both the goal I set for myself have been reached, since I reached the $1^{st}$ position of the leaderboard at the end of the competition and at the same time I deeply experimented with some of the most interesting machine learning techniques I studied during the course.

## 6. REFERENCES


[1] C. W. A. K. Muller and G. E. Birch. Linear and nonlinear methods for brain-computer interfaces. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 11(2), June 2003.

[2] A. Korik, N. Siddique, R. Sosnik, and D. Coyle. E3d hand movement velocity reconstruction using power spectral density of eeg signals and neural network. In *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 8103–8106, Aug 2015.

[3] D. J. McFarland, A. T. Lefkowicz, and J. R. Wolpaw. Design and operation of an eeg-based brain-computer interface with digital signal processing technology. *Behavior Research Methods, Instruments, & Computers*, 29(3):337–345, 1997.

[4] I. Walker. Deep convolutional neural networks for brain computer interface using motor imagery. September 2015.

[5] H. Zou and T. Hastie. Regularization and variable selection via the elastic net.