Advanced Machine Learning Algorithms "AdaBoost & Gaussian Process Classification"

Léonard Pasi
School of Engineering (STI)
École polytechnique fédérale de Lausanne (EPFL)
Lausanne, Switzerland
Email: leonard.pasi@epfl.ch

Jonas Lynge Vishart
School of Engineering (STI)
École polytechnique fédérale de Lausanne (EPFL)
Lausanne, Switzerland
Email: jonas.vishart@epfl.ch

***Abstract*—The measurement and analysis of Steady State Visually Evoked Potentials (SSVEPs) have recently become one of the common ways of working with electroencephalographic (EEG) signals and EEG research in general, due to its simple system structure, short training time and relatively low signal-to-noise ratio. This report aims to classify a SSVEP dataset from [1]. However, several signal processing methods and transformations must be applied in order to classify the raw EEG data. Among others, this report has only selected channels of interest, transformed the data into Power Spectral Densities (PSDs), and only keeping the frequencies of interest. Next, the reduced dataset was used as features in the classification made with AdaBoost and Gaussian Process Classifier (GPC). The optimal hyperparameters for AdaBoost and the optimal kernel for GPC were found through a grid search and evaluated upon their accuracy. The result was very similar train accuracies, achieved in very different ways.**

I. Introduction

A Brain-Computer Interface (BCI) system allows people to communicate through brain signals without the need of any muscular movement. These systems were mainly developed with the objective of creating possible communication methods for patients suffering from motor neuron disease. Recently, steady state visually evoked potentials (SSVEPs) have been widely used in visual BCIs due to advantages of high information transfer rate and little user training [2]. SSVEPs are signals that are natural responses to visual stimulation at specific frequencies. When the retina is excited by a visual stimulus ranging from 3.5 Hz to 75 Hz, the brain generates electrical activity at the same (or multiples of) frequency of the visual stimulus.

In 2017, Wang et al. presented a benchmark SSVEP dataset acquired with a 40- target BCI speller [1]. The dataset consists of 64-channel Electroencephalogram (EEG) data from 35 healthy subjects while they performed a cue-guided target selecting task. There are 40 stimulation frequencies ranging from 8 Hz to 15.8 Hz with an interval of 0.2 Hz. The stimulation duration in each trial was five seconds.

The goal of this project is to thoroughly compare two classifiers: the Gaussian Process classifier (GPC) and AdaBoost; we will do so using the aforementioned dataset. AdaBoost, short for Adaptive Boosting, is a statistical classification meta-algorithm that combines "weak learners" (such as decision stumps) by training them in a smart way: each learner complements the previous ones by focusing more on misclassified samples. AdaBoost was the first "boosting" algorithm, an approach to machine learning that has seen a tremendous amount of success. Tens of variants have been proposed in the last two decades, in order to reduce overfitting, increase robustness to noise or tailor it to specific problems. Here we will focus on the popular Real AdaBoost, which typically converges faster than the original algorithm, achieving a lower test error with fewer boosting iterations.

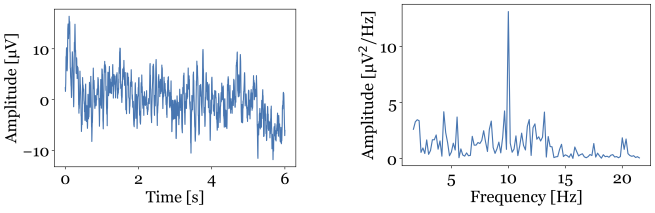
The Gaussian Process Classifier (GPC) is a non-parametric supervised machine-learning algorithm designed to solve classification problems. It is a Gaussian process (GP), hence a probability distribution over possible functions that fit the training data. The probabilistic characteristics is highly used within Gaussian Processes to give an uncertainty in predictions, nevertheless it will not be considered in this project. The main hyperparameter of the Gaussian process classifier is the choice of the Gaussian kernel that determines almost all the generalization properties of the GP model. The parameters of the model are often auto-tuned during model fitting, and estimated via maximum marginal likelihood, which is a huge advantage of GP as it makes it robust to the initial values of hyperparameters in a kernel [3]. Here we will consider various kernels, and select the one that is optimal for the classification of our dataset.

The rest of this report is organized as follows: in Section II, we describe the preprocessing of the dataset of interest, as well as the terms of comparison of the two classifiers and the adopted methods. The results presented in Section III are later discussed in Section IV. A brief conclusion is included in section V to close the report.

II. Methods

In machine learning projects, the raw data is often not in a form that is amenable to learning, but features that are can be constructed from it. This is typically where most of the effort goes. To some extent, this project was no different. The original dataset has the dimension: 35 x 64 x 1500 x 40 x 6, corresponding to 35 subjects, 64 electrode channels, 1500 time samples, 40 flicker frequencies and 6 iterations. Only 8 out of 40 flicker frequencies are considered: 8, 9, 10, 11, 12, 13, 14 and 15 Hz. This reduction in the number of classes, which also implies a reduction in the number of data points

and an increase of the inter-class variance, makes the classification problem much more simple (although not too simple). Furthermore, since SSVEP relies on visual information, only the channels placed at the visual cortex are of interest, and hence only 9 out of the 64 channels are kept, namely Pz, PO3, PO5, PO4, PO6, POz, O1, Oz and O2. As standard machine learning algorithms are usually not well suited to work on raw time series, we approach the problem in the frequency domain by computing the Power Spectral Density of each signal. This makes even more sense considering that each class corresponds to a specific flicker frequency which is expected to be found in the signals. A visualization of a given signal in time domain and its transformation in the frequency domain is given in Fig.1. Furthermore, as the nine electrodes are expected to record the same signals, the channels are averaged together after transformation in the frequency domain, thus reducing dimensionality and noise. Averaging before the transformation decreased the final performance. The mean power spectral density for each class is shown in Fig.2.



(a) (b)

Fig. 1: EEG data from the 1st subject, the 1st iteration, the Pz channel, and at the 10 Hz flicker frequency: in time domain (a) and in frequency domain as a power spectral density (b)

As Fig.2 suggests, what really enables to differentiate between classes are the first 2 harmonic peaks of each flicker frequency. Thus, we reduce the data points to 16-dimentional vectors by selecting the frequency components corresponding to the flicker frequencies and their double. This approach improves both accuracy and training speed, enabling extensive grid searches despite the limited time and processing power available. The final dimension of the dataset is 1680 x 16.

As was mentioned in the previous section, since it was first presented in 1996, tens of variations of AdaBoost have been proposed. In this project, we choose to focus on Real AdaBoost, which was introduced a few years later and typically achieves lower test error with fewer boosting iterations. The algorithm is implemented in the popular scikit-learn python library. In the rest of the report, we will use the terms "AdaBoost" and "Real AdaBoost" interchangeably. Although in principle boosting algorithms can be used with any classifier, traditionally, Adaboost was intended to combine weak base learners (such as decision stumps). However, it has been shown that it can also effectively combine strong base learners (such as deep decision trees), producing an even more accurate model. In this project, we will investigate whether these findings apply, by using decision trees as base learners

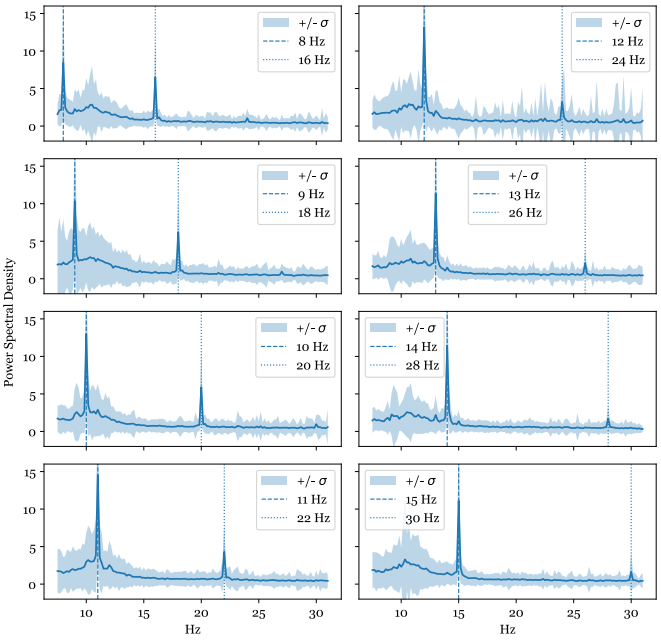


Fig. 2: Mean Power Spectral Density for each flicker frequency (i.e. for each class). Higher frequencies are left out of the plot.

and testing different depths: 1, 5 and 10. In addition to the type of base learner, AdaBoost has two main hyperparameters: the number of base learners and the learning rate. For each decision tree depth value, we will perform a grid search with standard 10-fold cross validation, with three repetitions. In other words, for each set of hyperparameters, the algorithm is trained and tested 30 times. The maximal number of decision trees will be adjusted for each considered depth in order to keep training times comparable across grid searches. It is often said that AdaBoost is prone to overfitting, thus we will look at both training and testing accuracy.

The Gaussian Process Classifier will also be implemented using scikit-learn python library. Among other things, it will be considered whether GPC is robust to the selection of hyperparameters as it is capable of auto-tuning its model hyperparameters during training. Nevertheless, a simple grid search will first be made to select the optimal kernel and later there will be looked into the auto-tuning.

On top of evaluating the performance of the classifiers, we also investigate their time complexity, both during training and testing. AdaBoost runs in $O(Nb \cdot f)$, where Nb is the number of base learners and f is the runtime of the weak learner in use. The construction of a decision tree through the CART algorithm (as in scikit-learn) runs in $O(N \cdot log(N) \cdot M)$, where N is the number of features and M is the number of data points. Unfortunately, this complexity doesn't take into account the maximal depth. Intuitively, there should be a linear dependency of the runtime with respect to the depth of the tree at training and testing both. In short, we expect AdaBoost's computational complexity to be $O(Nb \cdot N \cdot log(N) \cdot M)$ at training, without taking into account the trees' depth.

As for the Gaussian Process Classifier, it trains in $O(M^3)$. While we couldn't find any information about the testing time complexity, we expect it to be $O(M_{train} \cdot M_{test})$. As a non-parametric model, GPC needs to iterate over all training points when predicting the label of a new point, and this for every testing point. Finally, we should note that in general, the actual runtime analysis of an algorithm depends on the specific machine on which the algorithm is being implemented upon. Hence, we shall not come to rash conclusions from empirical plots.

Several papers suggest using SVM as classifier on SSVEP datasets. Consequently, SVM will be used as a baseline, while AdaBoost and Gaussian Process Classifier (GPC) will be used as main classifiers.

III. Results

Before presenting the classification results, the two classification algorithms (AdaBoost and GPC) and the baseline (SVM) are compared in terms of computational cost on a random generated dataset, with varying number of samples and number of features. The dataset is equally split into training and testing set. The classifiers were defined as follows:

1) AdaBoost with 50 decision stumps, and learning_rate=1.0 (named "V1" in the plots)
2) AdaBoost with 50 decision trees (with 5 levels), and learning_rate=1.0 (named "V2" in the plots)
3) GPC with an RBF kernel, and a length-scale $l = 4.0$
4) SVC with an RBF kernel, and a regularization parameter $C = 1.0$

The runtime complexity plots display the mean value, averaged over 5 iterations, with its standard deviation. Fig.4 and Fig.3 show the dependency over the number of samples and the number of features, respectively. The absolute value of the y-axis is not really meaningful, as the measured times are highly machine dependent.

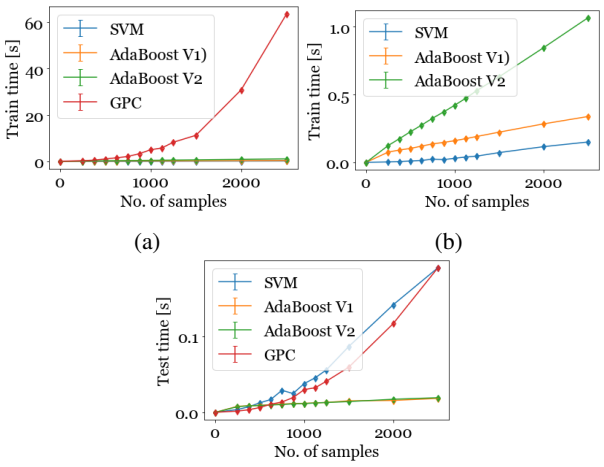


Fig. 3: Runtime complexity as a function of the number of samples during training (a)-(b) and during testing (c), with the number of features set at 20.

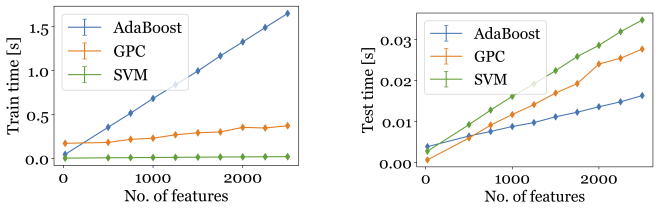


Fig. 4: Runtime complexity as a function of the number of features during training (a) and during testing (b), with 200 data points.

Fig.3 seems to confirm that GPC's computational training cost is cubic in the number of training points. Furthermore, testing time appears to be more than linear, although not necessarily quadratic as would be expected from the previous section (note that the number of testing samples equals the number of training samples). Perhaps surprisingly, SVM and GPC are very similar in terms of testing time, with SVM being even a bit slower. As for AdaBoost, at training it looks linear in the number of samples (as expected), and in the number of features (which is not expected).

A. Real AdaBoost

The results of the grid searches for Real AdaBoost are displayed in Fig.5,6,7. These plots reveal a number of insights. First, it appears that performance can be increased by using more complex base learners, with similar time complexity. Second, the generalization error of AdaBoost continues to improve by adding classifiers even after the training error has reached zero (see Fig.7, first row). This is a rather interesting finding and goes against the popular belief in machine learning, that given two classifiers with the same training error, the simpler of the two will likely have the lowest test error. Third, AdaBoost seems rather resistant to overfitting, although some traces of it can be found in the plots.

Using the time measurements taken during the grid search, the time complexity of AdaBoost with respect to its hyperparameters can be investigated empirically. The plots are displayed in Fig.8, 9, 10. As expected, neither the training nor the testing time depend on the learning rate (although Fig.9 seems to indicate either wise). Additionally, both training and testing times are linear in the number of estimators.

Finally, a confusion matrix is displayed in Fig.X for the best set of hyperparameters, which is found using the 10-level decision tree as base learner. While the model with (number of estimators, learning rate) = (800, 1.0) looks better at first, it is not statistically different from the simpler model (200, 1.0). Therefore, we choose the latter as the best Real AdaBoost model. Its accuracy is $89.9 \pm 2.1\%$. It trains in 4.62 ± 0.25 s on the training sets of the 10-fold cross validation, and it tests in 0.041 ± 0.0035 s on the corresponding testing sets.

B. Gaussian Process Classification

GPC is a kernel method, and it is therefore important to specify the kernel hyperparameter according to the dataset.

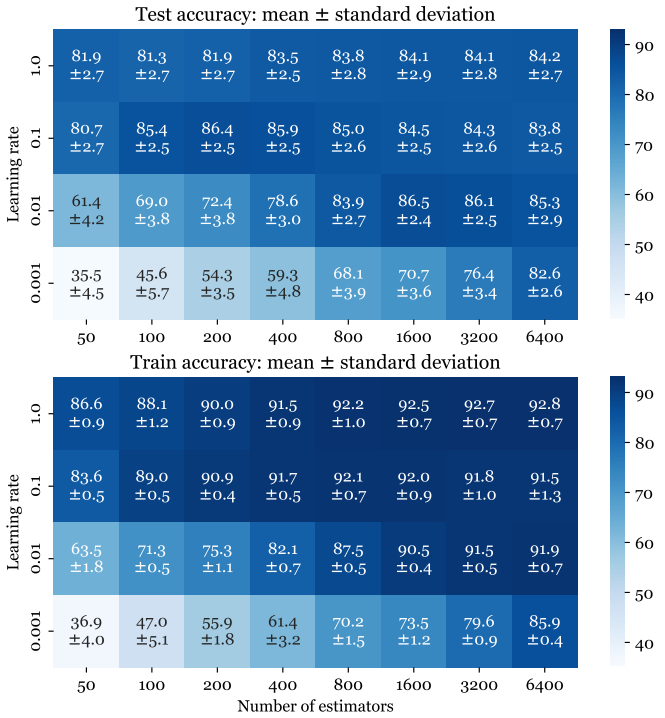


Fig. 5: Grid Search for Real AdaBoost over the number of base estimators and the learning rate, with decision stumps as base learners

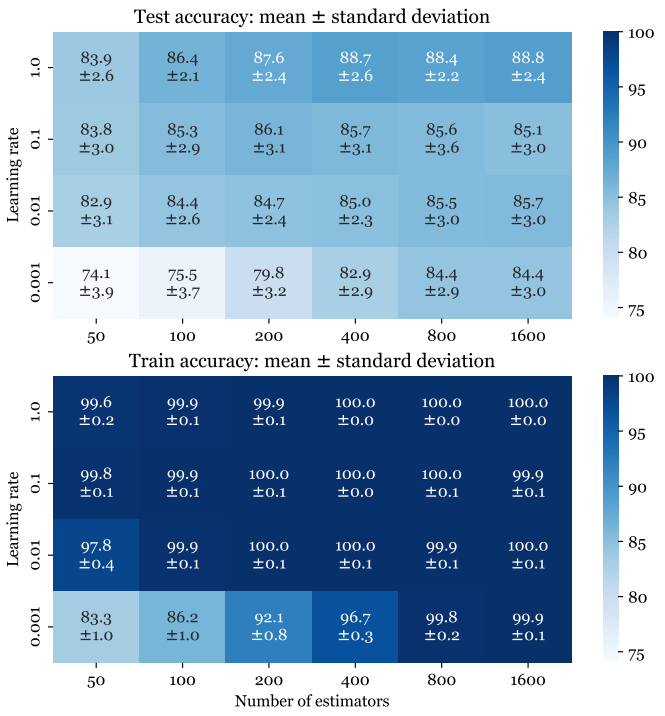


Fig. 6: Grid Search for Real AdaBoost over the number of base estimators and the learning rate, with 5-level decision trees as base learners

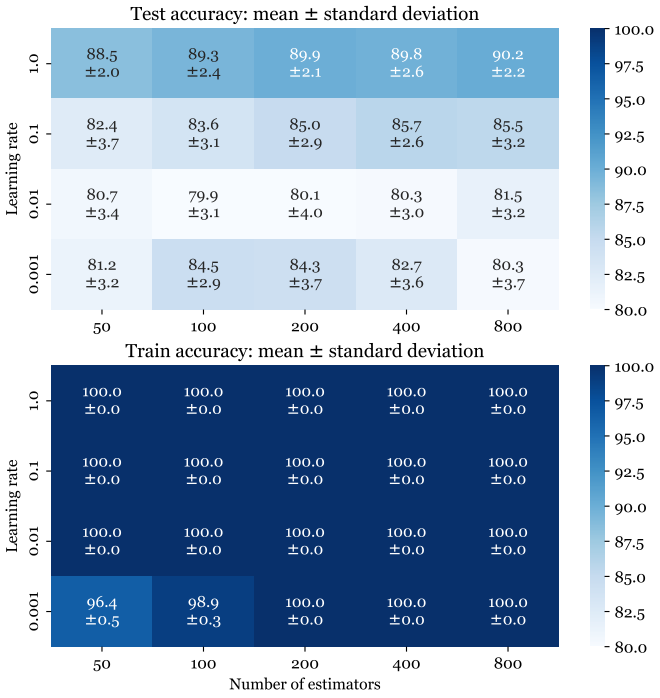


Fig. 7: Grid Search for Real AdaBoost over the number of base estimators and the learning rate, with 10-level decision trees as base learners

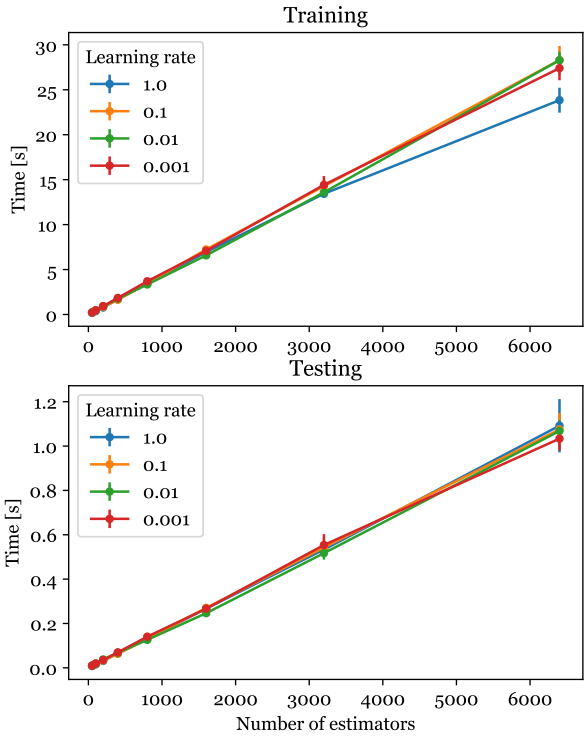


Fig. 8: Real Adaboost with decision stumps: time complexity as a function of hyperparameters

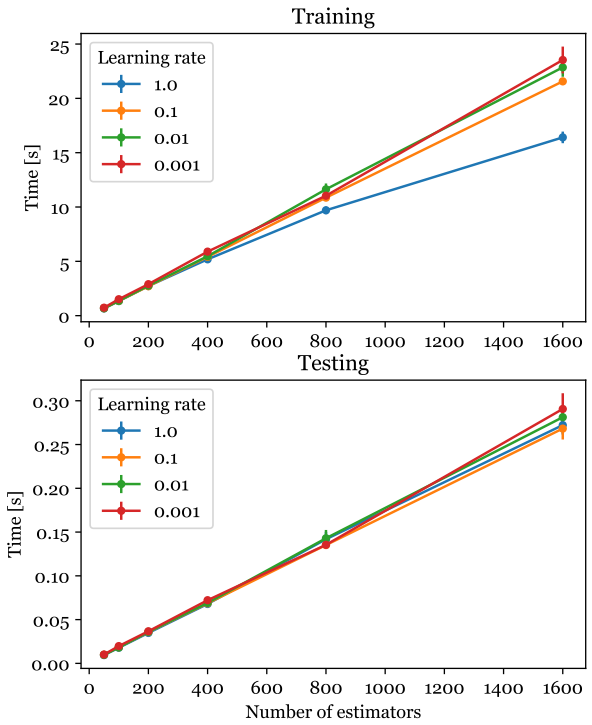


Fig. 9: Real Adaboost with 5-level decision trees: time complexity as a function of hyperparameters

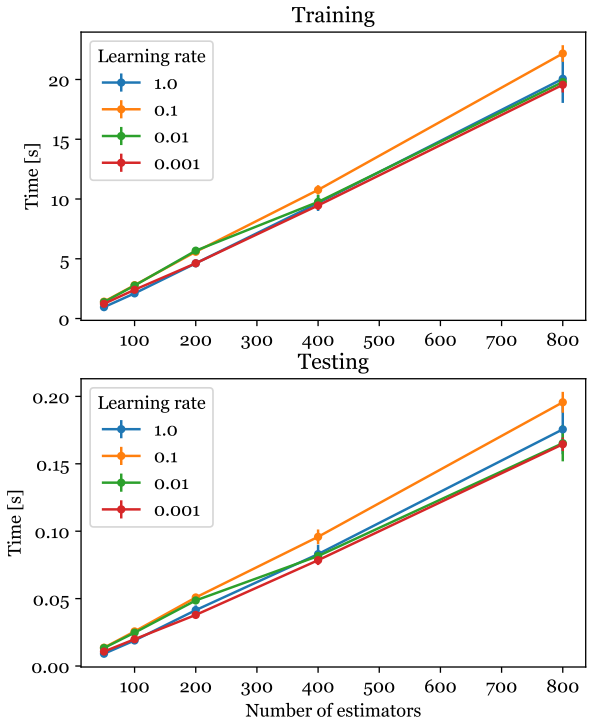


Fig. 10: Real Adaboost with 10-level decision trees: time complexity as a function of hyperparameters

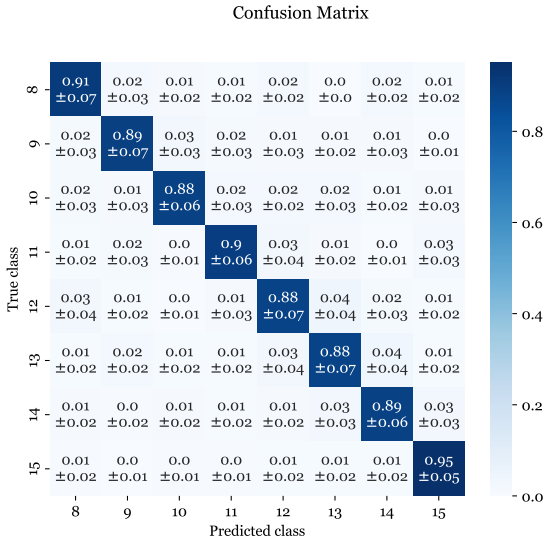


Fig. 11: Confusion Matrix for Real Adaboost with 200 10-level decision trees, and learning rate = 1.0. Means and standard deviations are computed through a 10-fold cross validation, with 3 repetitions. The matrix is normalized row-wise.

Typical kernels are RBF, DotProduct, Matern & RationalQuadratic. Some kernels are visualized in Figure 12 by using the details about Gaussian Processes kernels and their expressions found in Scikit-learn [4].

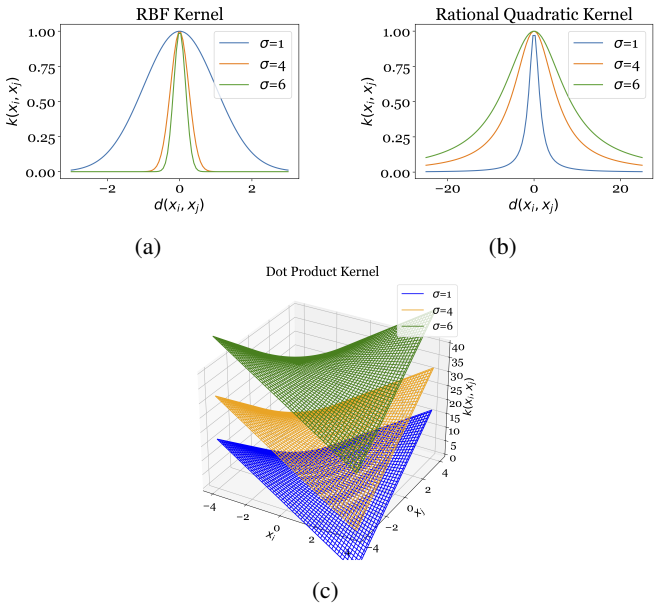


Fig. 12: Visualization of RBF kernel (a), RationalQuadratic kernel (b), and Dot Product kernel, with different values of σ.

A grid search was performed to compare different kernel types and length-scales. The results are seen in Table I using a 4-fold cross validation with 2 repetitions. Note that this is different from the number of cross validations and repetitions

used for AdaBoost. The accuracies should not be compared right away, since the different cross validations schemes result in different sizes of the training set.

Kernel	Train [%]	Test [%]
1*RBF(1.0)	60.9±6.7	58.8±4.4
1*RBF(4.0)	94.1±0.8	89.8±1.3
1*RBF(6.0)	94.2±0.8	89.7±1.3
1*DotProduct(1.0)	91.8±1.3	89.2±0.5
1*DotProduct(4.0)	91.8±1.3	89.2±0.5
1*DotProduct(6.0)	91.8±1.3	89.2±0.5
1*Matern(1.0)	72.1±5.4	69.6±4.9
1*Matern(4.0)	96.5±0.7	89.6±0.9
1*Matern(6.0)	96.5±0.7	89.6±0.9
1*RationalQuadratic(1.0)	95.4±0.4	89.4±0.9
1*RationalQuadratic(4.0)	95.4±0.4	89.4±0.9
1*RationalQuadratic(6.0)	95.4±0.4	89.4±0.9

TABLE I: Accuracies obtained by using Gaussian Process Classifier with different kernels.

The optimal kernel and configuration for GPC was found to be `1*RBF(4.0)` based on its high accuracy on the test set ($89.8 \pm 1.3\%$), and despite its relatively high standard deviation. Additionally, from the grid search, the GPC appears to highly depend on the choice of kernel and its parameters. While a `1*RBF(1.0)` led to a low accuracy of $60.9 \pm 6.7\%$ then `1*RBF(4.0)` resulted in the highest accuracy on the test set. Instead of the grid search, Scikit-learn provides the variable `n_restarts_optimizer` that allows the user to define a number of restarts in order to auto-tune and find the kernel parameters that maximize the log-marginal likelihood. The use of automatic tuning is expected to make the GPC robust and less sensitive to the choice of hyperparameters. However, a natural consequence of including the restarts is a longer computation time during training. In order to check the robustness of auto-tuning, some kernels from Table I were selected, which led to the results in Table II.

Kernel (initial hyperparameters)	Train [%]	Test [%]
1*RBF(1.0)	94.2±0.8	89.7±1.3
1*RBF(4.0)	94.2±0.8	89.7±1.3
1*Matern(1.0)	96.5±0.7	89.6±0.9
1*Matern(4.0)	96.5±0.7	89.6±0.9
1*RationalQuadratic(1.0)	95.4±0.4	89.4±0.9
1*RationalQuadratic(4.0)	95.4±0.4	89.4±0.9

TABLE II: Accuracies obtained by using Gaussian Process Classifier with different kernels, including `n_restarts_optimizer`=5 for auto-tuning of the hyperparameters of the model.

Table II shows that by including auto-tuning, the GPC is less sensitive to the initial choice of hyperparameters. It resulted in high accuracies for all the selected kernels. The auto-tuning finds an acceptable trade-off between complexity and accuracy. Nevertheless, a GPC with the initial parameters `1*RBF(4.0)` will be used in the further analysis. This choice of kernel and kernel parameters was further tested in a 10-fold cross validation (same as for AdaBoost) with no repetitions. The overall testing and training accuracies are $90.2 \pm 2.1\%$ and $93.9 \pm 0.3\%$, respectively. A clearer picture can be obtained from a confusion matrix, given in Fig.13. The confusion matrix shows a general high accuracy, above 87%, for all classes.

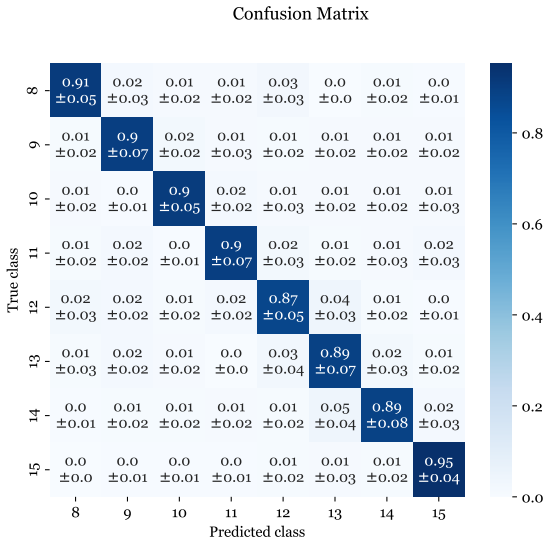


Fig. 13: Confusion matrix for GPC with `1*RBF(4.0)` to evaluate the (test) accuracy. The means and standard deviations are computed through a 10-fold cross validation, with no repetitions. The matrix is normalized row-wise.

IV. Discussion

Both classifiers result in reasonably accurate models: $89.9 \pm 2.1\%$ (or $90.2 \pm 2.2\%$, if we increase the number of estimators) for AdaBoost and $90.2 \pm 2.1\%$ for GPC. However, they do so in different ways. First, we notice the large difference in training accuracy: $100 \pm 0\%$ for AdaBoost and $93.9 \pm 0.3\%$ for GPC. Second, there is a huge difference in training times: 4.62 ± 0.25s for AdaBoost and 135 ± 13 s for GPC. This makes GPC quite painful to use: on the original 40-class dataset (with 5 times more training points), we couldn't even run it. The difference in testing time is also significant: 0.041 ± 0.0035 s for AdaBoost and 0.60 ± 0.06 s for GPC. Note that these measurements are taken using the same computer. Third, despite the high training time, it should be noted that good performance was obtained much faster with GPC, especially if using the automatic optimization of hyperparameters, while it took quite an extensive grid-search to achieve the same accuracy with AdaBoost.

The confusion matrices in Fig.11 and 13 both showed that the classifiers performed worse on some classes. For example, the 12 and 13 Hz flicker frequency were "more misclassified" with other classes. For this reason, Figure 14 has looked into some misclassifications of the test set, from the 13 Hz class.

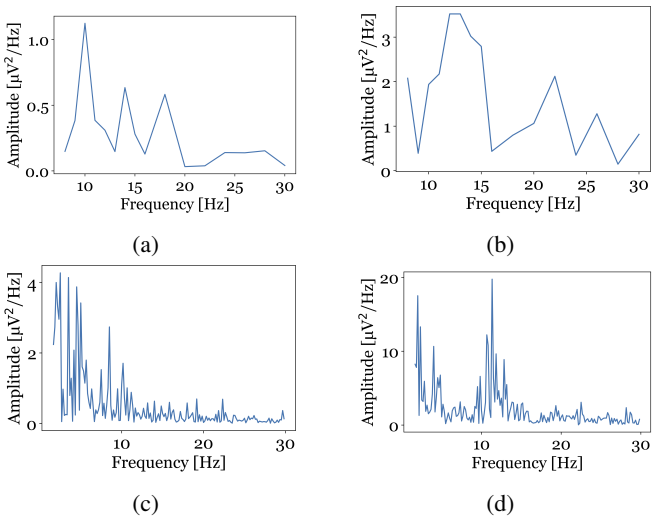


Fig. 14: Four figures that each show a 13 Hz flicker frequency signal, taken from the test set. After classification (a) was classified as 9 Hz, and (b) as 14 Hz. Figure (c) and (d) are the signal of (a) and (b), respectively, before signal was further reduced from 751 to 16 data points.

In Figure 14 (a), the 13 Hz flicker frequency was classified as 9 Hz. There are several reasons why this particular case is indeed a difficult classification task; 1) the largest peak appears around 9-10 Hz, and 2) a 2nd peak appears at 18 Hz, which supports the periodicity and hence the classification as 9 Hz. Those noisy peaks appearing in the periodicity of the actual flicker frequencies are very difficult to avoid or even filter out. A popular signal preprocessing technique known as filter banks might not even fix this issue. A filterbank is a "bank" of filters and can be created as a set of sine-cosine signals that matches the stimulation frequencies. In addition, Figure 14 (b) of the 13 Hz signal was also misclassified as 14 Hz. It shows many noisy peaks around 10-15 Hz, but also in the following frequencies. Nevertheless, we see peaks at both 13 Hz and 26 Hz as expected. Overall, the classification algorithms result in well and reasonable classifications. However, if higher accuracies are to be obtained, additional signal processing techniques must be implemented or we would have to look into other classification algorithms. Several papers suggest using Canonical Correlation Analysis (CCA) often in combination with SVM. CCA is a multivariate statistical method that finds the correlation between two sets of data, and can be used in SSVEP recognition by finding the correlation to a set of sine-cosine reference signals that matches the stimulation frequencies [5]. Finally, it would be of interest to compare to other classification algorithms such as deep neural networks that are capable of handling large amount of data and find patterns even in complex datasets and signals.

V. Conclusion

In this project, a SSVEP dataset was classified with the machine learning algorithms Real AdaBoost and Gaussian Process Classifier. The algorithms were evaluated in terms of hyperparameters, computational cost and performance. The challenge with model optimization is to improve the test accuracy while keeping the time cost as low as possible. Therefore, this project considered both the computational costs as function of number of samples, dimension of dataset, and hyperparameters. The huge advantage of GPC was its ability to auto-tune the model parameters during fitting, leading to generally high predictions, however with the cost of extra computation time during training (and testing). Without auto-tuning, the GPC was found to train in $O(M^3)$, and hereby way more demanding than SVM and AdaBoost that both showed a linear time cost as a function of samples. The optimal kernel for GPC, found through a simple grid search, was `1*RBF(4.0)`, while the optimal AdaBoost algorithm was with 200 10-level decision trees and a learning rate of 1.0. The test accuracy was 89.9% using AdaBoost and 90.2% using GPC. It is of interest to achieve even higher accuracies, and multiple signal preprocessing steps can still be implemented. Applying filter banks is one option, but generally the application of filters cannot remove the noisy peaks appearing at the exact flicker frequencies. Other approaches as increasing the amount of data will only lead to higher computational costs. Consequently, it might be necessary and interesting to look into and compare to other classification algorithms such as deep neural network that is more capable of handling large datasets.

References

[1] Yijun Wang et al. "A Benchmark Dataset for SSVEP Based Brain Computer Interfaces". In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 25.10 (2017), pp. 1746–1752. DOI: 10.1109/TNSRE.2016.2627556.

[2] M. Neghabi, H. R. Marateb, and A. Mahnam. "Comparing Steady-State Visually Evoked Potentials Frequency Estimation Methods in Brain-Computer Interface With the Minimum Number of EEG Channels". In: *Basic and clinical neuroscience* 10(3) (2019), pp. 245–256. DOI: https://doi.org/10.32598/bcn.9.10.200.

[3] Zexun Chen and Bo Wang. "How priors of initial hyperparameters affect Gaussian process regression models". In: *Neurocomputing* 275 (2018), pp. 1702–1710. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2017.10.028. URL: https://www.sciencedirect.com/science/article/pii/S092523121731679X.

[4] Scikit-Learn. *Gaussian Processes*. https://scikit-learn.org/stable/modules/gaussian_process.html, Retrieved 2022-05-30.

[5] Valeria Mondini et al. "Sinc-Windowing and Multiple Correlation Coefficients Improve SSVEP Recognition Based on Canonical Correlation Analysis". In: *Computational Intelligence and Neuroscience* 2018 (2018). DOI: https://doi.org/10.1155/2018/4278782.