hyperSPACE: Automated Optimization of Complex Processing Pipelines for pySPACE

Torben Hansing, Mario Michael Krell, Frank Kirchner*
Robotics Research Group
University of Bremen
Robert-Hooke-Str. 1, 28359 Bremen
hansa@informatik.uni-bremen.de

Abstract

Even though more and more algorithms are introduced for optimizing hyperparameters and complex processing pipelines, it still remains a cumbersome task for domain experts. Grid search is still used in most cases despite its deficiencies. In this paper, we combine the optimization library Hyperopt with the signal processing and classification environment pySPACE to completely automatize the optimization process. Even though no preliminary knowledge is required, interfaces for domain and algorithm experts were added to accelerate the optimization. As a proof of concept, the new framework is applied to electroencephalographic data which requires exhaustive optimization of a complex processing pipeline.

1 Introduction

The capability to process complex input signals is a vital property of future "never to turn off" robots that will continuously operate in highly dynamic environments to perform their missions and tasks over long periods of time. This kind of long term autonomy decisively requires the capability to adapt to changes both in the environment as well as in the system itself. Wear and tear in the robots mechanics are a consequence of physical existence and must be compensated on the algorithmic side. Apart from online learning, it is also necessary that robots can optimize and reconfigure their processing. This holds for other applications of autonomous systems and for complex data processing chains as they are used in brain-computer interfaces as well.

But tuning signal processing and classification has been a complex task for now. The numerous algorithms used to pre-process, classify, or otherwise process the input data have many so called *hyperparameters*. These parameters need to be optimized to achieve state of the art performance with a processing pipeline. Additionally, the choice of the algorithms that should be used within a pipeline has a great impact on the performance as well. So far, these problems have been mainly solved by experts in combination with the simple grid search for decades. But, in the past few years, major progress has been done to automatically optimize hyperparameters and structure.

One simple extension of grid search is (an asynchronous) pattern search, which has been used for example to optimize a SVM jointly with the hyperparameters [5]. In pattern search, a grid around a center is defined (pattern) and evaluated. If a new optimum is found, the center is shifted. Otherwise, the pattern is scaled to analyze a smaller area around the center. Other optimization approaches are random search [2], Bayesian optimization [4, 8, 13] especially sequential model-based optimization [6], deep learning [1, 11], and many more.

A very mature optimization approach is the Tree-of-Parzen-Estimators algorithm [3] which is public available in the Hyperopt package [2]. This package has been combined with scikit learn [12] to

*Frank Kirchner is also affiliated to the Robotics Innovation Center, DFKI GmbH, Robert-Hooke-Str. 1, 28359 Bremen

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

enable hyperparameter and structure optimization in Hyperopt-sklearn [9]. A similar approach is Auto-WEKA [15]. Unfortunately, it still holds for both approaches that they use a handcrafted search domain. Furthermore, because the *WEKA* and *scikit-learn* libraries do not implement them, they do not yet include the whole data processing pipeline into the search domain as it is for example required for time series analysis. Time series analysis often uses temporal filtering, spatial filtering, feature generation, normalization, and classification/regression algorithms.
Recently, the signal processing and classification environment pySPACE [10] has been introduced. It provides interfaces to numerous processing algorithms and allows to construct complete processing pipelines from the unsegmented time series to the final probability output. Furthermore, it does not require much programming expertise, is easy to set up, and supports parallelization of large scale evaluations. Hence, it is suitable to process complex input data like electroencephalographic (EEG) data, that are time continuous, high dimensional, and have a low noise-to-signal ratio.
In this paper, we fuse Hyperopt and pySPACE and present an intuitive interface to optimize complex processing pipelines. This interface does not need any additional handcrafting but sets up everything automatically. Even though no expert knowledge is required, interfaces are provided to improve the optimization. For example, the algorithm expert can define general preferences for parameters in form of distributions, whereas the domain expert can reduce the search space by (de-)selecting certain algorithms. In contrast to the already existing approaches, our approach does not merge the different search domains, to keep the search domain tractable. It is very large, because of the possibilities to combine the numerous considered algorithms (about 300) and respective hyperparameters (9 in average) would result in a search domain of about 10^{22} combinations.

2 Method

This Section presents our approach to hyperparameter optimization of complex processing pipelines. In contrast to existing approaches, we create the pipeline search domain and the search domain for the hyperparameters for arbitrarily complex pipelines dynamically. Our approach is based on pySPACE to feature a larger variety of algorithms and to allow the creation of complex processing pipelines.
Since pySPACE features approximately 300 algorithms and a complex processing pipeline uses many different (i. e. more than five) algorithms, not all possible combinations can be tested. Hence, the search space is reduced by two kinds of experts. An **algorithm expert** is a person, who implements a processing algorithm of any kind and / or has further knowledge about it and thus, is able to define which input data types an algorithm can process and which output data types it produces. This knowledge is already implemented in pySPACE and can be used to reduce the combinations that need to be tested during the optimization process. In a valid processing pipeline the output data type of each algorithm matches with the input data type of the next algorithm. For further reduction of the search space, the **domain expert** can include and / or exclude certain algorithms because they are not useful for optimization (e.g. debugging or visualization nodes) or not appropriate for the given domain (e.g. time filtering algorithms for already pre-filtered data).
After generating the processing pipelines, that need to be optimized, the hyperparameter spaces for them need to be constructed as well. Even though defaults have been defined in hyperSPACE, we expect an **algorithm expert** to define the underlying distribution that should be assumed to make a good choice for a hyperparameter. Note, that the distributions only have to be very rough estimates and that a good documentation should already provide this information. Our approach makes this, mostly hidden, knowledge more explicit and can be seen as a normal part of documenting algorithms. Thus, we adopted the distributions described by Bergstra et al. [2]. So far, we support 10 distributions:

1. no optimization – The default parameter should usually not be changed, like for example the convergence criterion of a support vector machine (SVM).
2. (weighted) choice – The parameter provides some categorical choice where priorities can be set. For the kernel of an SVM for example, the linear kernel is appropriate in 50% of all cases, the RBF kernel in 25%, and the remaining weights can be distributed equally.
3. unified sampling – Any value within an interval can be chosen like for example choosing ν for the ν-SVM in $[0.1, 0.9]$.
4. q-unified sampling – Any value within an interval can be chosen with a resolution of q.
5. log-unified sampling – Any value within the logarithmic scaled interval can be chosen.

6. q-log-unified sampling – Combination of 4. and 5. The maximal number of iterations of a SVM can be expressed using q-log-uniform distribution with a minimum of 1, a maximum of ∞ and a q of 1.
7. normal sampling – The probability of choosing a value is described by a normal distribution. For example a weighting factor in $[0, 1]$ could be defined with $m = 0.5$ and $\sigma = 0.2$ and extreme cases outside of the boundaries have to be projected.

8.-10. See 4,5,6 but for normal instead of unified distribution

For a given, feasible processing pipeline, we provide Hyperopt with the respective performance values for hyperparameter settings sampled by it in sequential order, whereas the evaluation usually considers several evaluation parts that can be executed in parallel. Apart from this interface, the feasible processing pipelines are constructed in parallel and they are optimized independently from each other. Building a search space over all algorithm combinations would result in a structure that becomes too large to handle. Furthermore, the optimization can start while the pipeline search space is still being explored. This is crucial for very large search spaces. Last but not least, a search space for every processing pipeline enables the user to store and even reuse the optimization results for similar input data in an iterative optimization process.

3 Evaluation

3.1 Data and Processing

For evaluation, we used electroencephalographic data acquired by Kirchner et al. [7]. While playing a virtual labyrinth game, the subject had to respond to certain rare stimuli by pressing a buzzer whereas no reaction was required to another more frequent stimuli. The resulting pattern to the relevant stimulus in the EEG signal is an event-related potential called P300. Our objective is to distinguish the P300 pattern from the other pattern which comes from the frequent stimulus and which is very similar to the pattern, when the important stimulus was not perceived.
We used the suggested processing pipeline as baseline [7]. For speeding up the optimization, we relied on the given segmentation, standardization, subsampling to 25 Hz, and low-pass filtering to 4Hz, which also fits to the signal characteristics that were analyzed. The further processing consisted of a spatial filter (xDAWN) with 8 filters, a fitting of local straight lines to obtain the incline as a feature, a feature standardization, and a SVM classification with an optimization of the decision threshold on the training data, where the SVM was fixed with a class weighting of $1 : 3$, and the regularization parameter was optimized with a 5-fold cross-validation ($10^0, 10^{-1}, \ldots, 10^{-6}$).

3.2 Experiments and Evaluation Scheme

We performed 5 different experiments with each reducing domain knowledge and increasing the search domain. This also increases the danger of overfitting to the validation data:

1. We optimized the 10 hyperparameters of the classifier (SVM).
2. All 20 hyperparameters were optimized but the structure of the baseline was kept.
3. The optimizer chose all hyperparameters and the classifier.
4. The feature generation was optimized, in addition to the classifier.
5. All the processing was optimized (except for the temporal preprocessing). In this case, irrelevant algorithms were removed by an algorithm and domain expert to reduce the search space and priorities were set according to the baseline.

Figure 1 shows our scheme for the overall evaluation. The first three runs of the second recording session of one subject have been used to optimize (1 vs.1 comparison) the pipeline with hyperSPACE and to train the processing pipeline with the optimized parameters. The remaining data are used for three different evaluations. The objective of those different schemes is to determine, how far the optimized pipeline generalizes for unseen data with different difficulty stages concerning the transfer. The first evaluation is done on the remaining two runs from the same session. This evaluation shows, whether the processing pipeline is able to classify very similar, but unseen data as well. The second

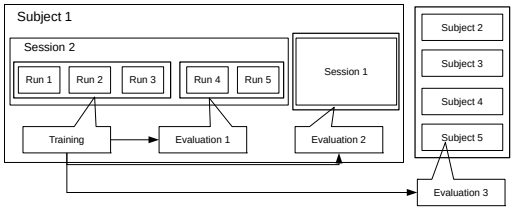

Figure 1: Evaluation Scheme

Exp.	Eval.1	Eval.2	Eval.3	Valid.
Basel.	82.6	71.2	65.4	
Exp.1	84.6	73.1	73.5	87.1
Exp.2	86.2	66.7	61.4	87.9
Exp.3	85.2	74.4	66.7	87.7
Exp.4	87.3	70.5	62.7	87.8
Exp.5	66.3	51.6	50.1	87.8

Table 1: Evaluation results of the experiments as the averages of % balanced accuracy

evaluation is done on another session of the same subject, since slight differences in behavior and cap setup are expected. In the last experiment, we evaluate on all recordings from the remaining subjects. Due to class imbalance, we report the arithmetic mean of true positive/negative rate, called balanced accuracy, for all evaluations [14].

3.3 Results and Discussion

Table 1 shows the average performance of the processing pipelines of all 5 experiments (Exp.) and the baseline (Basel.) for the 3 evaluation cases (Eval.) and the optimized value achieved on the training data (Train.). Exp. 5 had to be interrupted after approx. 22% of the processing but at that time, most processing pipelines (based on 32 algorithms with 29121 resulting pipelines) had already converged. First of all, it can be seen that our framework is capable of optimizing processing pipelines. Especially when all hyperparameters are optimized instead of optimizing only the SVM parameters, the optimized value (Train.) increases to a value around 87.8% balanced accuracy. A constant increase with the degree of freedom cannot be observed properly, due to random initialization and sampling of the optimization process. For the evaluation on the same session, the optimized pipelines outperform the baseline (except for Exp. 5) which shows the necessity of optimizing pipelines and that hyperSPACE can be used for that.
Unfortunately, in many cases the performance is lower than the baseline. For Exp. 5, where everything was optimized, but also for the transfer of the pipeline to a different session / subject, overfitting of the optimization approach on the validation data in the optimization evaluation can be observed. For compensation, larger training and validation data sets have to be used for the optimization, which would largely increase the processing workload. Here, we sticked to a smaller setup to analyze the convergence and transferability. Additionally, we only analyzed the optimization of one subject but further subjects could be analyzed to reduce the factor of randomness in the optimization process.

4 Conclusion

In this paper, we introduced a new approach to optimize complex processing pipelines. Our approach is capable of handling a much larger number of algorithms (provided by pySPACE) and requires no handcrafted search domain. In contrast to existing approaches, it works completely automatic. It largely benefits from a new "documentation approach" for algorithm developers and provides an interface for domain experts to reduce the search domain. Our evaluation showed in one challenging use case, that the presented approach is improving performance in contrast to the classical approach. Nevertheless, a reduction of the search domain is largely recommended when the processing resources are limited (time, processors), or to avoid overfitting to the validation dataset during the optimization.
Future steps are the application to other domains, the extension to other optimization strategies, nested optimizations, the distinction between general and dataset specific parameters, and the interface to a database to benefit from intermediate results. Furthermore, a more exhaustive evaluation is required with a larger dataset and other optimization approaches like random search.

References

[1] Y. Bengio. Learning Deep Architectures for AI. *Foundations and Trends® in Machine Learning*, 2(1):1–127, 2009. ISSN 1935-8237. doi: 10.1561/2200000006.

[2] J. Bergstra, D. Yamins, and D. D. Cox. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *12th Python in science conference (SCIPY 2013)*, 2013.

[3] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2546–2554. Curran Associates, Inc., 2011.

[4] K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. Hoos, and K. Leyton-Brown. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *NIPS workshop on Bayesian Optimization in Theory and Practice*, pages 1–5, 2013.

[5] T. Eitrich and B. Lang. Efficient optimization of support vector machine learning parameters for unbalanced datasets. *Journal of Computational and Applied Mathematics*, 196(2):425–436, nov 2006. ISSN 03770427. doi: 10.1016/j.cam.2005.09.009.

[6] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential Model-based Optimization for General Algorithm Configuration. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization*, LION'05, pages 507–523, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-25565-6. doi: 10.1007/978-3-642-25566-3_40.

[7] E. A. Kirchner, S. K. Kim, S. Straube, A. Seeland, H. Wöhrle, M. M. Krell, M. Tabie, and M. Fahle. On the applicability of brain reading for predictive human-machine interfaces in robotics. *PLoS ONE*, 8(12):1–19, 12 2013. doi: 10.1371/journal.pone.0081732.

[8] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter. Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets. may 2016.

[9] B. Komer, J. Bergstra, and C. Eliasmith. Hyperopt-sklearn: Automatic hyperparameter configuration of scikit-learn. *ICML 2014 AutoML Workshop*, 2014.

[10] M. M. Krell, S. Straube, A. Seeland, H. Wöhrle, J. Teiwes, J. H. Metzen, E. A. Kirchner, and F. Kirchner. pyspace - a signal processing and classification environment in python. *frontiers in Neuroinformatics*, 7, December 2013.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[13] J. Snoek. *Bayesian Optimization and Semiparametric Models with Applications to Assistive Technology*. PhD thesis, University of Toronto, 2013.

[14] S. Straube and M. M. Krell. How to evaluate an agent's behaviour to infrequent events? – Reliable performance estimation insensitive to class distribution. *Frontiers in Computational Neuroscience*, 8(43):1–6, jan 2014. ISSN 1662-5188. doi: 10.3389/fncom.2014.00043.

[15] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proc. of KDD-2013*, pages 847–855, 2013.