Expanding NIRS Auto ML (NAML) to Better Facilitate Neural Data Analysis

A Major Qualifying Project
Submitted to the Faculty of Worcester Polytechnic Institute
In partial fulfillment of the requirements for the
Degree in Bachelor of Science
In Computer Science
By

Ellery Buntel

Date: 12/08/2020
Project Advisors:

Professor Erin Solovey, Advisor
Professor Rodica Neamtu, Advisor

Abstract

NIRS Automated Machine Learning (NAML), is a system that automates the running of time series machine learning algorithms on neural data [1]. This project expanded NAML by adding seven new algorithms, implementing manifold validation, and giving the user the ability to balance their datasets. NAML 2.0 gives more precise results than its predecessor and can still be utilized without expertise in machine learning. NAML 2.0 will allow researchers to classify brain states quickly, easily, and consistently. NAML 2.0 has been benchmarked on neural data for both accuracy and speed.

Table of Contents

List of Figures

List of Tables

Introduction

Over the last few years, time series data analysis techniques have become far more advanced. State of the art algorithms are both efficient and accurate. However, the complexity of these algorithms has also increased significantly. This presents labs that focus on time series analysis with a problem: advanced training in programming and data analysis is required to use these modern algorithms. This can cause a bottleneck in the research process as only advanced researchers are capable of effectively analyzing time series data. NAML 1.0 [1] was created to help solve this problem by allowing users of any skill level access to machine learning data analysis algorithms. NAML 1.0 provided a powerful framework for running these algorithms, offering users four choices in terms of algorithms.

In this MQP report, we describe NAML 2.0, which expands on NAML 1.0's capabilities to genuinely allow for state-of-the-art analysis. NAML 2.0, like NAML 1.0, uses the sktime [2] time series machine learning library to implement the algorithms it offers. Sktime is an open source Python library that builds upon more commonly used machine learning libraries like sklearn [11] to provide a resource for analysts studying time series datasets. By using this library, we can reduce the risk of implementation errors, and ensure that NAML 2.0's algorithms run consistently. Since the creation of NAML 1.0, sktime has been updated with new algorithms including two multivariate algorithms, MrSEQL and MUSE. Adding these algorithms has made NAML 2.0 a more robust and versatile system.

<u>Background</u>

Goals of this Project -

The development of NAML 2.0 was focused on accomplishing the following three goals:

1. <u>Compatibility with updated sktime library and bug fixes</u>

NAML 1.0 was rendered non-functional by updates to sktime's file structure. Therefore, the first goal of this project was to update NAML 1.0's import system to reflect sktime's new structure. Additionally, the multivariate shapelet transform method included in NAML 1.0 was removed from sktime so replacing it with a working multivariate method was also a priority. Finally, rather than use NAML 1.0's test-train split method which could mistakenly produce empty datasets, sklearn's manifold validation method was implemented.

2. <u>Addition of new algorithms</u>

The second goal of this project was to extend the capabilities of the NAML system by adding new classifiers and methods to it. NAML 1.0 included four classifiers available, and, after updates to sktime, did not have a usable multivariate classification method. To remedy this, seven new classification methods were added to NAML 2.0. Included among these new methods are NAML 2.0's only multivariate classifiers, MrSEQL and MUSE, as well as algorithms that had not been implemented in sktime at the time of NAML 1.0's development.

- NAML 1.0 algorithms:
    - Shapelet transform (Multivariate)
    - K Nearest Neighbors (Univariate)
    - Proximity Forest (Univariate)
    - Time series Forest (Univariate)
- NAML 2.0 algorithms:

- MrSEQL (Multivariate)
- MUSE (Multivariate)
- K Nearest Neighbors (Univariate)
- Proximity Forest (Univariate)
- Time series Forest (Univariate)
- WEASEL (Univariate)
- Bag of SFA Symbols Ensemble (Univariate)
- Elastic Ensemble (Univariate)
- Temporal Dictionary Ensemble (Univariate)
- RISE (Univariate)

3. Implementation of new features

Finally, NAML 1.0 had neither a validation method nor a method for balancing the data. These are commonly used to improve the validity of classification results by controlling for anomalies through manifold validation, and class balancing through over- and undersampling. These features will allow users to better analyze unbalanced datasets and should give users a more accurate idea of how their classifiers are performing.

New Algorithms -

Bag of SFA Symbols Ensemble (BOSSE) [3]:

The BOSSE algorithm classifies time series data by first extracting patterns from the time series, then it reduces noise by filtering the extracted patterns, and finally it compares those filtered patterns for differences. The extraction of patterns is done using Symbolic Fourier Approximation (SFA) [4] which transforms segments of the data into strings based on the data in that segment. It is these strings that are filtered and then compared by the BOSSE algorithm. During the SFA transformation, a sliding window is used to create strings for each part of the dataset. The BOSSE algorithm tries multiple window sizes and uses the best one.

Elastic Ensemble [5]:

The Elastic Ensemble algorithm takes multiple distance measures using different distance equations like DTW (Dynamic Time Warping) and ensembles them together to create a classifier that is more accurate than one created with just one distance measure.

MrSEQL [6]:

MrSEQL is a multivariate time series classification method that classifies based on a symbolic representation of the time series. This representation is created using a symbolic fourier approximation which reduces the data from a full timeseries to a series of symbols selected by the algorithm. These strings of symbols are then compared to classify the datasets. MrSEQL aims to both be more easily interpretable and less resource intensive than equivalent classification algorithms. It is also unique in that it creates multiple symbolic representations of each subsection of the dataset rather than just one. During classification, the algorithm selects the best symbolic representation available.

Random Interval Spectral Forest (RISE) [7]:

RISE is an algorithm included in the HIVE-COTE ensemble classifier. The HIVE-COTE ensemble classifier combines many classification methods, and is considered to have state of the art accuracy. RISE is one such classification method, but is implemented in sktime as a standalone algorithm. RISE creates multiple decision trees and then ensembles them together. Each tree is created by first extracting features from a random interval of the dataset and then using these features to create the decision tree.

Temporal Dictionary Ensemble (TDE) [8]:

The Temporal Dictionary Ensemble algorithm uses a sliding window along with a symbolic fourier approximation to convert the time series into a series of words. Then, the TDE algorithm forms a dictionary by counting the number of occurrences of a word in each time series. Finally, classification is run on the dictionaries representing each time series.

Word ExtrAction for time SEries cLassification (WEASEL) [9]:

WEASEL efficiently classifies large time series datasets by transforming the time series into feature vectors with a sliding window. These feature vectors are then classified. It is not as robust as some ensemble classifiers, but it is considerably faster.

MUSE [10]:

MUSE is a multivariate classifier that uses WEASEL to transform the time series into a multivariate feature vector. This vector is then reduced in size as feature selection removes non-discriminative features from each dimension of the time series. Finally, the vector is analyzed by a classifier to output a result.

Methodology

Validation -

NAML 2.0 implements manifold validation using one of sklearn's [11] implementations of manifold validation [12]. Manifold validation is an effective way of eliminating a level of uncertainty from the accuracy results of each algorithm. This helps prevent situations where the user is presented with an accuracy that is deceptively high or low. First, the data is split into multiple folds; this split is done to ensure that no time series is subdivided. After the data has been split, the system iteratively excludes each part from the training set and instead uses it as the test set. Each result is recorded and the mean of the results is returned to the user. In particular, NAML 2.0 implements stratified k-fold validation [12]. Stratified k-fold validation is a type of manifold validation that attempts to keep the class ratio in each fold as similar to the original dataset's class ratio as possible. This ensures that no fold will be without examples of each class which helps prevent outlier results.

Univariate and Multivariate Algorithms -

NAML 2.0 implements both univariate and multivariate time series classification algorithms. As fNIRS data is generally multivariate, multivariate algorithms were a priority in development. However, sktime does not support many multivariate algorithms so the univariate concatenation, or column ensemble transformations are often required.

Univariate concatenation appends each time series to the end of the previous time series. This creates a univariate time series that univariate classification algorithms can be run on; this results in a loss of information. Each row in a multivariate time series represents a different measurement being taken in conjunction with other measurements in the data. In some cases these measurements are correlated because they share a time. However, when you append each time series to the end of the last, you lose this correlation. This method is functional, but others are better for classification.

Column ensemble allows the user to define individual classifiers for each row of the data. After each classifier finishes, they are all combined into a single result that is presented to the user. This allows the user to effectively classify a multivariate dataset using only univariate algorithms.

Alternatively, users can make use of NAML 2.0’s two multivariate classification algorithms: MUSE and MrSEQL.

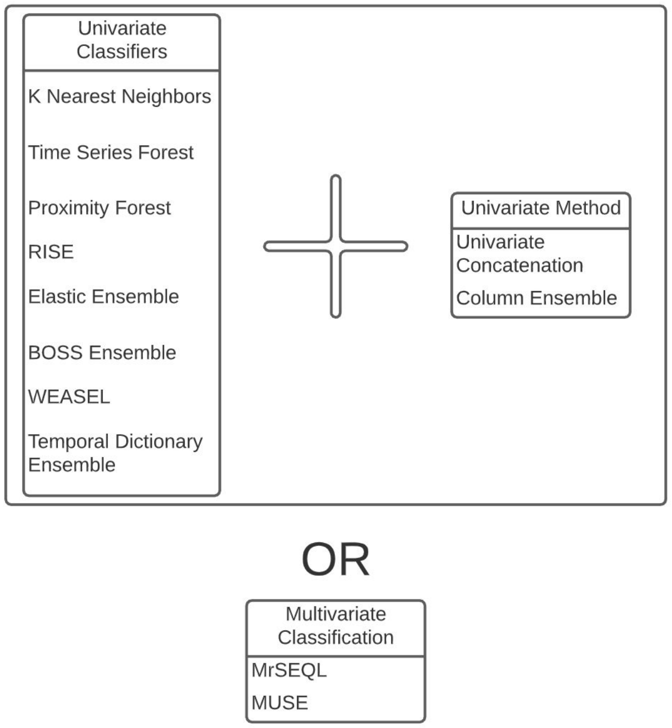


Figure 1: Univariate / multivariate methods chart

Class Balancing -

NAML 2.0 balances classes by oversampling the minority class after the dataset has already been split into folds for validation. Another Python library, Imbalanced-learn [13], is used to oversample the data. This oversampling is done by including copies of time series - that have the minority class label - in the training set. The training set with oversampled data is then run through the classifier as normal.

Technical Evaluation -

NAML 2.0's technical evaluation had two goals:

1. To get a baseline accuracy and time for each algorithm. The baseline accuracies and times will be helpful for future users of NAML 2.0 to compare against and should give us clues as to which algorithm will be most effective for different purposes.
2. To examine how class balancing and dataset size affects classifier accuracy and efficiency.

Note: At the time of the technical evaluation the Elastic Ensemble and Proximity Forest algorithms were not functioning in the latest version of sktime. I have left these algorithms available in NAML 2.0 in the hope that they will work in the future, but they could not be included in the technical evaluation.

Technical Evaluation

This technical evaluation explores how a user might utilize NAML to classify different brain states. This evaluation is also a benchmark for classifier performance and speed. This technical evaluation is based on experiments run on our lab's AX-CPT (AX- Continuous Performance Task) dataset. This dataset consists of neural data recorded using an fNIRS [14] device while the participant was engaged in the AX-CPT [15] cognitive control task. There are two classes in the data: AX wrong response, and AX correct response. AX wrong response occurs when a participant makes a mistake in their task, and AX correct response occurs when they correctly carry out the task.

Full AXCPT Dataset for Participants 209, 214, and 216

The dataset consists of 10,800 rows with 92 columns each. Each data point is the reading from each of 21 sensors placed on the participants' head. The dataset is imbalanced as 80% of the events are AX correct responses and only 20% of events are AX wrong response. 5-fold stratified cross validation was used. NAML classified each participant's dataset only once. The computer used to conduct the experiments has 16 gigabytes of RAM, and an 8-core Intel i7-6700HQ processor.

Algorithm	**Accuracy**	**Time (seconds)**
MrSEQL	77.78%	1126.95
MUSE	53.7%	4857.43
TSF	62.22%	40.52
KNN	52.22%	1.62
RISE	67.41%	1043.5
BOSSE	58.52%	134307.22
WEASEL	64.81%	15155.23

TDE	67.3%	19476.5

Table 1: Full AXCAXWR Dataset Experiment Results

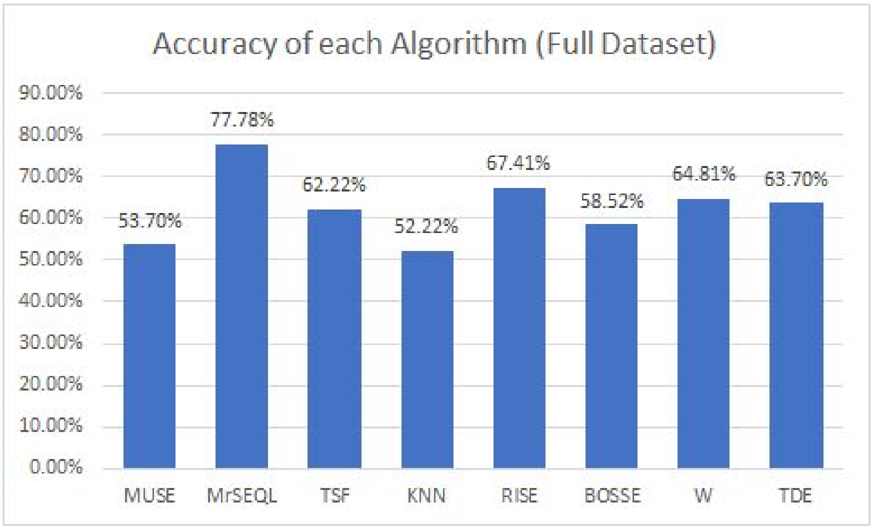


Figure 2: Full Experiment Accuracies

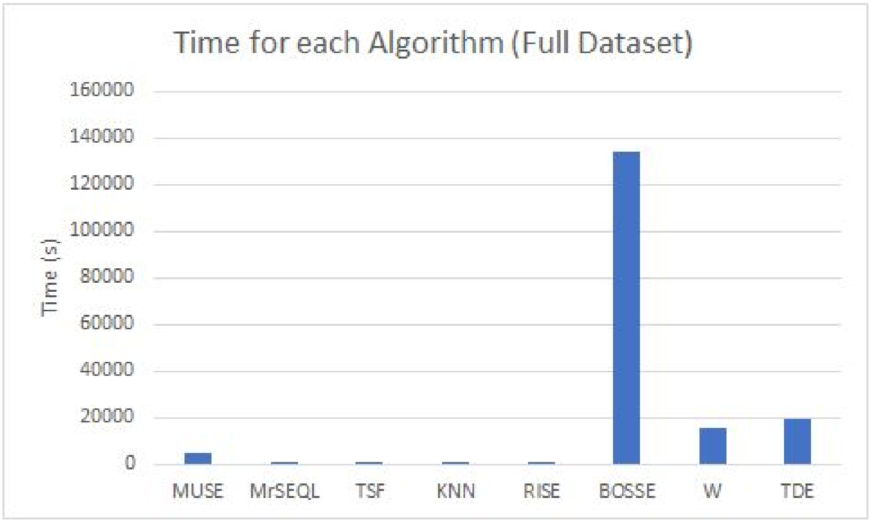


Figure 3: Full Experiment Times

AXCPT AXCAXWR Dataset for Participant 209

The dataset consists of 3,720 rows with 92 columns each. Each data point is the reading from each of 21 sensors placed on the participants' head. The dataset is imbalanced as 76.3% of the events are AX correct responses and only 23.7% of events are AX wrong response. 5-fold stratified cross validation was used. NAML classified each participant's dataset three times. The accuracies and times given below are the averages of each run. The computer used to conduct the experiments has 16 gigabytes of RAM, and an 8-core Intel i7-6700HQ processor.

Algorithm	**Balancing**	**Average Accuracy**	**Average Time (seconds)**
MrSEQL	True	76.39%	331.08
MrSEQL	False	76.37%	358.41
MUSE	True	66.41%	932.23
MUSE	False	63.82%	735.72
TSF	True	86.03%	15.32
TSF	False	84.13%	15.23
KNN	True	82.44%	1.14
KNN	False	83.45%	1.16
RISE	True	83.24%	439.51
RISE	False	85.67%	413.2
BOSSE	True	87.8%	23107.26
BOSSE	False	87.37%	23186.4
WEASEL	True	74.95%	774.94
WEASEL	False	75.26%	696.68
TDE	True	86.7%	2028.67
TDE	False	87.88%	2427.54

Table 2: AXCAXWR Participant 209 Dataset Experiment Results

Note: (OS) in the graphs below stands for oversampled

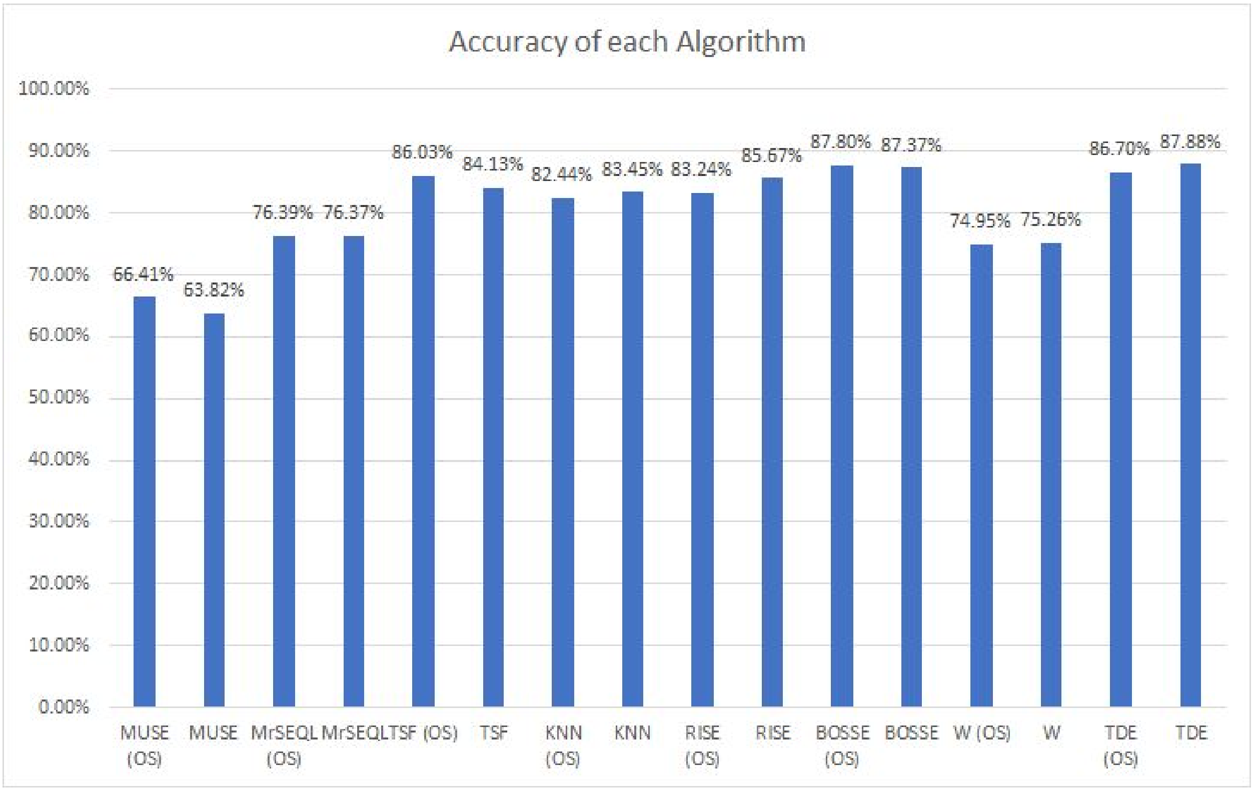


Figure 4: Single Participant Experiment Accuracies

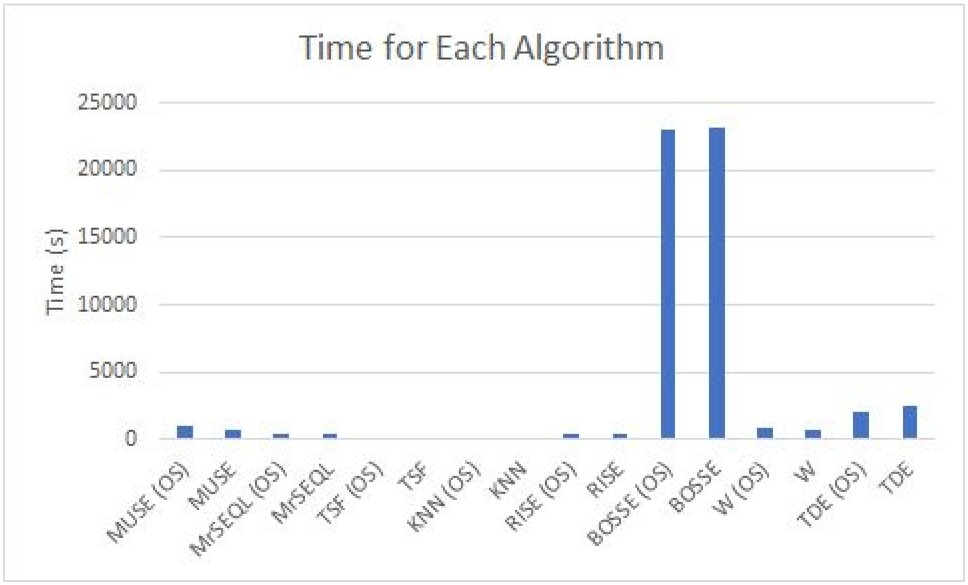


Figure 5: Single Participant Experiment Times

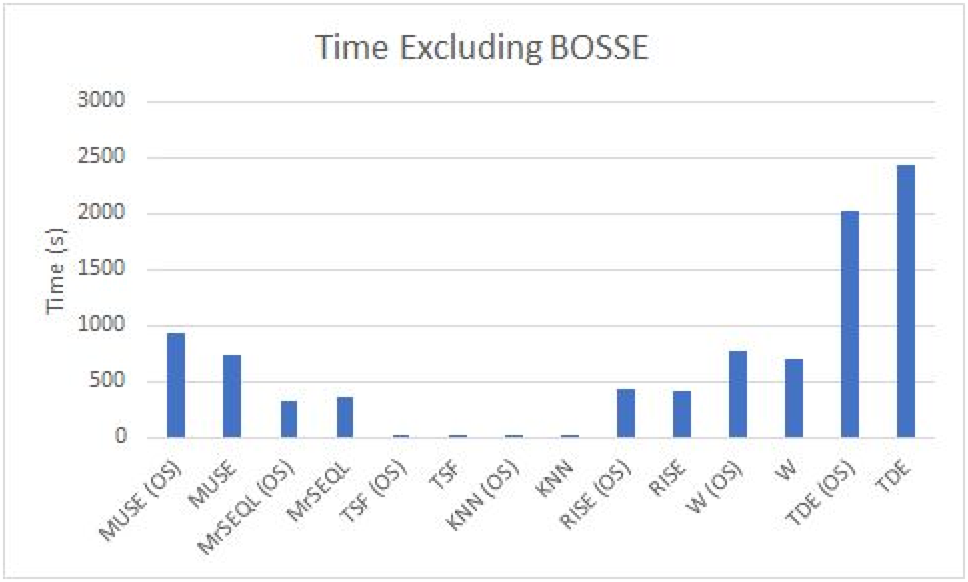


Figure 6: Single Participant Experiment Times Excluding BOSSE

Technical Evaluation Conclusions

The first thing to note is the difference in accuracies between the full experiment and the single participant experiment. The single participant experiment achieved significantly higher accuracies than the full dataset experiment. This is likely due to the inclusion of data from multiple participants. Including data from multiple participants increases the difficulty of classification as most participants' data is not very similar. While MrSEQL still achieved acceptable accuracy on this dataset most of the algorithms did not.

Another interesting feature of the data is that algorithm time doesn't increase linearly with the size of the dataset. The full dataset is about three times the size of the single participant dataset, but the algorithms took more than three times longer to run through the full dataset than the single participant dataset. This is true for most of the algorithms, but of particular note is BOSSE. BOSSE took about six and a half hours to run through the single participant dataset, but took 37 hours to run through the full dataset. The full dataset took BOSSE almost six times longer to complete than the single participant dataset. This suggests to me that users should carefully curate the size of their datasets before running them through NAML to avoid algorithms running for too long. On the other hand, KNN took almost the same amount of time on the full dataset as the single participant dataset. Therefore KNN might be a good choice when dealing with a large dataset.

Finally, the effects of balancing the dataset were different for each algorithm. While some, like MUSE and TSF, reached higher accuracies on average with balanced datasets, other algorithms, like KNN and RISE, actually got lower accuracies with balanced datasets. Balancing didn't have a uniform effect on the algorithm completion times either.

Overall, the data from these experiments allows us to clearly distinguish which algorithms are the best for each dataset. In particular, in the first experiment, the results show that MrSEQL is the best algorithm. This demonstrates that future NAML 2.0 users will be able to run their own experiments to find the best classifiers for their own datasets. While the results from experiment two are somewhat less conclusive, they do show that NAML provides multiple classifiers that can produce accurate results.

Future Work

This project has significantly expanded upon the capabilities of NAML 1.0, but there are other improvements that may be worth including in a future update.

The most important future improvement is adding new algorithms as they are implemented in sktime. NAML 2.0 implements most of sktime's classification algorithms at this point, but sktime is constantly being updated. Keeping up to date with sktime's algorithms will give NAML2.0 users access to new and potentially more effective algorithms. This should be the first priority of any subsequent versions of NAML.

Another source of new algorithms for NAML 2.0 is sktime-dl [16]. sktime-dl is an extension library for sktime that implements deep learning algorithms for time series datasets with TensorFlow / Keras. Deep learning algorithms have been used to great effect in analyzing brain data [17] so NAML 2.0 could benefit greatly from their inclusion. Making sktime-dl algorithms usable in NAML 2.0 would likely not require dramatic changes to the structure of the code. However, many deep learning methods use GPUs, which would increase the complexity of implementation.

Another area that could be improved is NAML 2.0's validation system. Though cross validation was added to NAML as a part of this project, there are many types of validation that haven't been included in NAML yet. Giving users the option to choose between multiple forms of validation would allow for even greater control over the results. sktime doesn't include validation methods at the time of writing, but sklearn has implementations of many validation methods including leave one out, shuffle split, and time series split. All of these validation methods could be made available to NAML users with minimal modifications to the code.

Finally, NAML 2.0 could benefit from an automatic parameter tuning system. Many of NAML 2.0's algorithms perform very differently based on the parameters inputted into them. This is why NAML 2.0 allows the user to set parameters in their config files. However, typical users who do not have data analysis experience will likely find tuning these parameters quite difficult. Those users in particular would benefit from a system that could determine the best parameters to use for each dataset. Manual implementation of such a system would be quite difficult, but sklearn has methods that may be applicable to NAML 2.0. At the time of writing, sktime has a parameter tuning cross validation method for time series forecasters, but none for classifiers.

Works Cited

1. Ikram, Fareya. NirsAutoML: Building an automated classification platform for fNIRS data. (2019).
2. Löning, Markus, Bagnall, Anthony, Ganesh, Sajaysurya, Kazakov, Viktor, Lines, Jason , and Király, Franz J.. "sktime: A unified interface for machine learning with time series." *arXiv preprint arXiv:1909.07872* (2019).
3. Schäfer, Patrick. The BOSS is concerned with time series classification in the presence of noise. Data Mining and Knowledge Discovery. 29. 10.1007/s10618-014-0377-7. (2015).
4. Schäfer, P., Högqvist, M.: SFA: a symbolic fourier approximation and index for similaritysearch in high dimensional datasets. In: EDBT. ACM (2012)
5. Lines, J., Bagnall, A. Time series classification with ensembles of elastic distance measures. Data Min Knowl Disc 29, 565–592 (2015). https://doi.org/10.1007/s10618-014-0361-2
6. Le Nguyen, Thach, et al. "Interpretable time series classification using linear models and multi-resolution multi-domain symbolic representations." Data Mining and Knowledge Discovery 33.4 (2019): 1183-1222.
7. Lines, Jason, Taylor, Sarah, and Bagnall, Anthony. Time Series Classification with HIVE-COTE: The Hierarchical Vote Collective of Transformation-Based Ensembles. ACM Trans. Knowl. Discov. Data 12, 5, Article 52 (July 2018), 35 pages. DOI:https://doi.org/10.1145/3182382
8. Middlehurst, Matthew, et al. "The Temporal Dictionary Ensemble (TDE) Classifier for Time Series Classification." The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases. (2020).

9. Schäfer, Patrick and Leser, Ulf . Fast and Accurate Time Series Classification with WEASEL. In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management (CIKM '17). Association for Computing Machinery, New York, NY, USA, 637–646. (2017). DOI:https://doi.org/10.1145/3132847.3132980
10. Schäfer, Patrick, and Leser, Ulf . "Multivariate time series classification with WEASEL+ MUSE." arXiv preprint arXiv:1711.11343 (2017).
11. Pedregosa, Fabian, Varoquaux, Gaël, Gramfort, Alexandre, Michel, Vincent, Thirion, Bertrand, Grisel, Olivier, Blondel, Mathieu et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.
12. Kohavi, Ron. "A study of cross-validation and bootstrap for accuracy estimation and model selection." In Ijcai, vol. 14, no. 2, pp. 1137-1145. (1995).
13. Lemaître, Guillaume, Nogueira, Fernando, and Aridas, Christos K.. "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning." The Journal of Machine Learning Research 18, no. 1 (2017): 559-563.
14. A. Serwadda et al., "fNIRS: A new modality for brain activity-based biometric authentication," 2015 IEEE 7th International Conference on Biometrics Theory, Applications and Systems (BTAS), Arlington, VA, (2015), pp. 1-7, doi: 10.1109/BTAS.2015.7358763.
15. Polizzotto, NR, Hill-Jarrett, T, Walker, C, Cho, RY. Normal development of context processing using the AXCPT paradigm. PLoS ONE 13(5): e0197812. (2018). https://doi.org/10.1371/journal.pone.0197812
16. Large, James, Bostrom, Aaron, Ismail Fawaz, Hassan, Löning, Markus et al. *sktime-dl* (0.1.0). Python Library. Large, James et al, (2019).

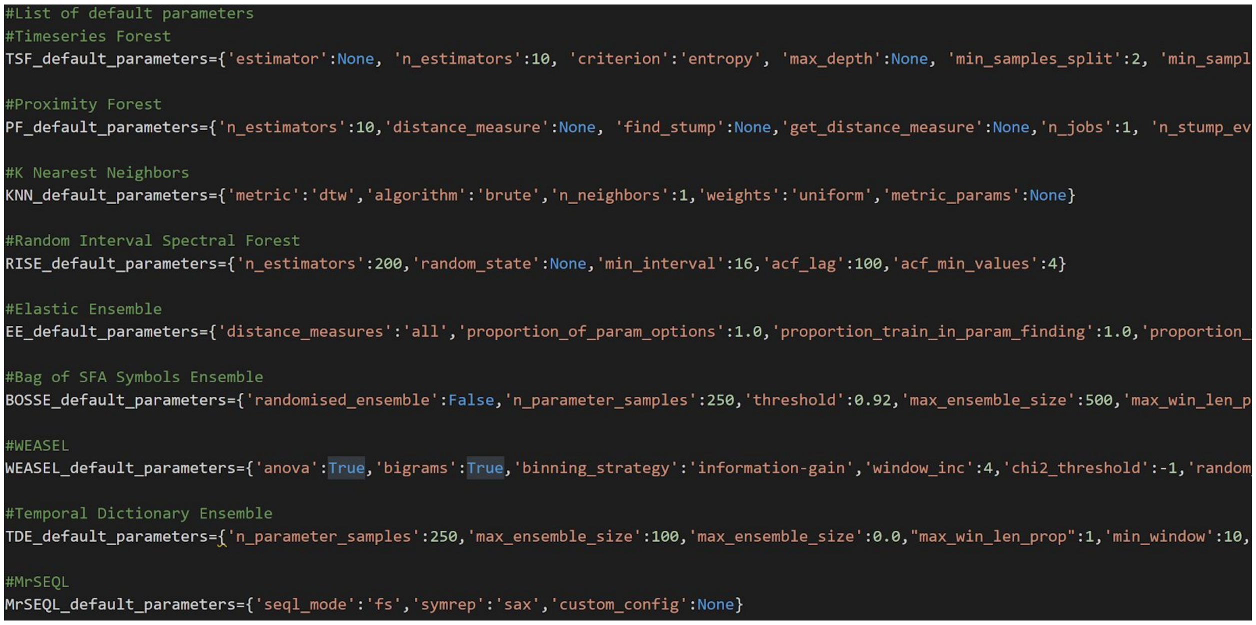
17. Jiang, L., Stocco, A., Losey, D.M. et al. BrainNet: A Multi-Person Brain-to-Brain Interface for Direct Collaboration Between Brains. Sci Rep 9, 6115 (2019). https://doi.org/10.1038/s41598-019-41895-7

Appendix A - Developer's Guide

In this section I will give guidance to developers who may work on NAML 2.0 in the future. These instructions are accurate at the time of writing (Fall 2020) but changes to the structure of NAML 2.0 itself, or to the structure of the underlying library, sktime, may render them inaccurate in the future.

Adding New Univariate Algorithms to NAML 2.0 -

1. Add the algorithm's default parameters to the list of default parameters at the beginning of naml.py. You do not need to add all of the algorithm's parameters, but users will only be able to change the ones inputted here in the config file. Below is a picture of the list of parameters near the top of naml.py.

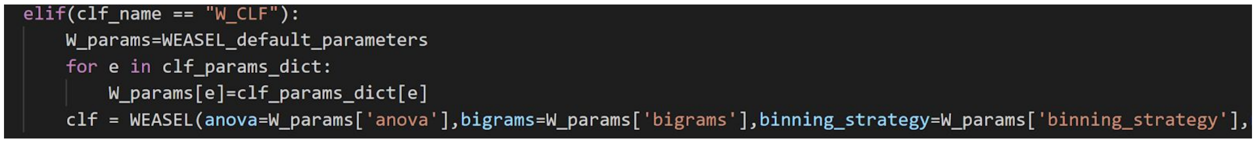


```
#List of default parameters
#Timeseries Forest
TSF_default_parameters={'estimator':None, 'n_estimators':10, 'criterion':'entropy', 'max_depth':None, 'min_samples_split':2, 'min_sampl

#Proximity Forest
PF_default_parameters={'n_estimators':10,'distance_measure':None, 'find_stump':None,'get_distance_measure':None,'n_jobs':1, 'n_stump_ev

#K Nearest Neighbors
KNN_default_parameters={'metric':'dtw','algorithm':'brute','n_neighbors':1,'weights':'uniform','metric_params':None}

#Random Interval Spectral Forest
RISE_default_parameters={'n_estimators':200,'random_state':None,'min_interval':16,'acf_lag':100,'acf_min_values':4}

#Elastic Ensemble
EE_default_parameters={'distance_measures':'all','proportion_of_param_options':1.0,'proportion_train_in_param_finding':1.0,'proportion_

#Bag of SFA Symbols Ensemble
BOSSE_default_parameters={'randomised_ensemble':False,'n_parameter_samples':250,'threshold':0.92,'max_ensemble_size':500,'max_win_len_p

#WEASEL
WEASEL_default_parameters={'anova':True,'bigrams':True,'binning_strategy':'information-gain','window_inc':4,'chi2_threshold':-1,'random

#Temporal Dictionary Ensemble
TDE_default_parameters={'n_parameter_samples':250,'max_ensemble_size':100,'max_ensemble_size':0.0,"max_win_len_prop":1,'min_window':10,

#MrSEQL
MrSEQL_default_parameters={'seql_mode':'fs','symrep':'sax','custom_config':None}
```

Figure 7: NAML 2.0 Default Parameters List

2. Add the algorithm to the classifierBuilder function. The if statement checks the name of the classifier requested in the config file and returns an instance of the classifier itself. To add a new algorithm simply add another elif section, create an instance of the

classifier, and give it the parameters inputted by the user. Below is an example of WEASEL's section of classifierBuilder.



```
elif(clf_name == "W_CLF"):
    W_params=WEASEL_default_parameters
    for e in clf_params_dict:
        W_params[e]=clf_params_dict[e]
    clf = WEASEL(anova=W_params['anova'],bigrams=W_params['bigrams'],binning_strategy=W_params['binning_strategy'],
```

Figure 8: WEASEL Section of classifierBuilder

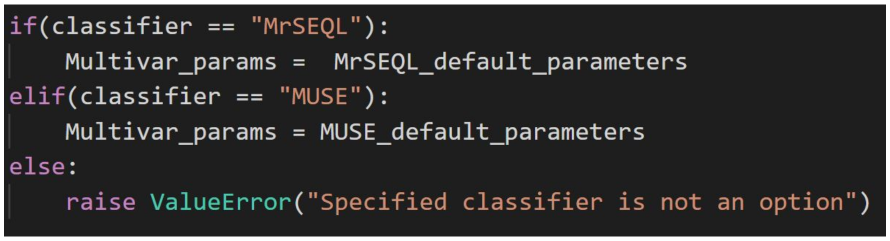
3. The algorithm should now be usable.

Note: NAML 2.0's algorithms all take data in the same format (as a Numpy array). If you want to add an algorithm that requires a different format then you will need to modify the concatenateMethod function and the columnEnsembleMethod function in addition to the above steps.

Adding New Multivariate Algorithms to NAML 2.0 -

Currently, NAML 2.0 supports two multivariate algorithms: MUSE and MrSEQL. Adding new multivariate algorithms is a similar process to adding new univariate algorithms, but there are a few key differences I will outline below.

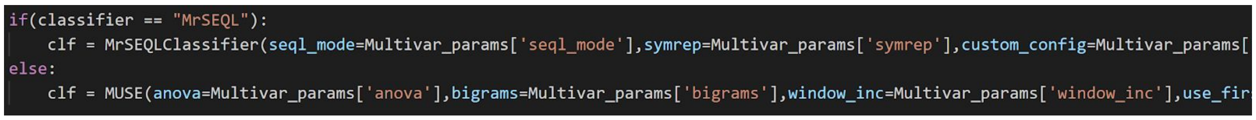
1. Add the algorithm's default parameters to the list of default parameters at the beginning of naml.py. This is exactly the same as in step one of the "Adding New Univariate Algorithms to NAML 2.0" guide.
2. Add the new algorithm to the MultivariateClassification method. First, add an elif statement to the section of code below that sets *Multivar_params* equal to the default parameter dictionary you created in step one.



```
if(classifier == "MrSEQL"):
    Multivar_params =  MrSEQL_default_parameters
elif(classifier == "MUSE"):
    Multivar_params = MUSE_default_parameters
else:
    raise ValueError("Specified classifier is not an option")
```

Figure 9: Multivar_params Code Section

Next, add an elif statement to the if statement shown below. In the body of the statement set *clf* to be an instance of your new classifier. If you want the user to be able to use their own parameters for your new algorithm, then set each parameter equal to *Multivar_params['name_of_parameter']*.



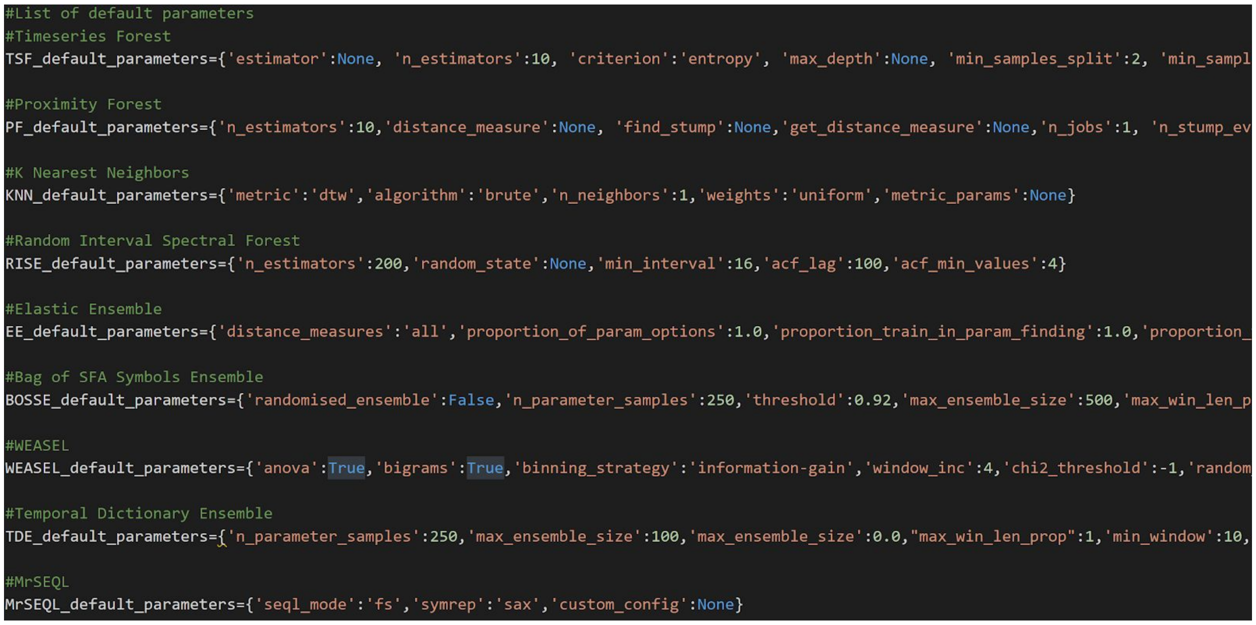
```
if(classifier == "MrSEQL"):
    clf = MrSEQLClassifier(seql_mode=Multivar_params['seql_mode'],symrep=Multivar_params['symrep'],custom_config=Multivar_params[
else:
    clf = MUSE(anova=Multivar_params['anova'],bigrams=Multivar_params['bigrams'],window_inc=Multivar_params['window_inc'],use_fir
```

Figure 10: Multivariate Classifier Instantiation Code Section

3. The algorithm should now be usable.

Changing Default Parameters for Algorithms -

The default parameters for each algorithm are found at the top of the naml.py file. These values are used when the user does not specify parameters of their own. If you want to change these default values then modifying just these lines will suffice. Below is a picture of the section.



```
#List of default parameters
#Timeseries Forest
TSF_default_parameters={'estimator':None, 'n_estimators':10, 'criterion':'entropy', 'max_depth':None, 'min_samples_split':2, 'min_sampl

#Proximity Forest
PF_default_parameters={'n_estimators':10,'distance_measure':None, 'find_stump':None,'get_distance_measure':None,'n_jobs':1, 'n_stump_ev

#K Nearest Neighbors
KNN_default_parameters={'metric':'dtw','algorithm':'brute','n_neighbors':1,'weights':'uniform','metric_params':None}

#Random Interval Spectral Forest
RISE_default_parameters={'n_estimators':200,'random_state':None,'min_interval':16,'acf_lag':100,'acf_min_values':4}

#Elastic Ensemble
EE_default_parameters={'distance_measures':'all','proportion_of_param_options':1.0,'proportion_train_in_param_finding':1.0,'proportion_

#Bag of SFA Symbols Ensemble
BOSSE_default_parameters={'randomised_ensemble':False,'n_parameter_samples':250,'threshold':0.92,'max_ensemble_size':500,'max_win_len_p

#WEASEL
WEASEL_default_parameters={'anova':True,'bigrams':True,'binning_strategy':'information-gain','window_inc':4,'chi2_threshold':-1,'random

#Temporal Dictionary Ensemble
TDE_default_parameters={'n_parameter_samples':250,'max_ensemble_size':100,'max_ensemble_size':0.0,"max_win_len_prop":1,'min_window':10,

#MrSEQL
MrSEQL_default_parameters={'seql_mode':'fs','symrep':'sax','custom_config':None}
```

Figure 11: NAML 2.0 Default Parameters List (Identical to Figure 7)

Appendix B - Config File Construction

Config File Values -

filePath: Relative path to the data file you wish to use.

loggingEnabled: If true will output logs as the program progresses.

targetCol: The name of the column of your data you wish to classify by.

percentTrain: The percentage of the initial dataset to be set aside for use as a testing set. This is only used if you choose the COLUMN_ENSEMBLE method.

Jobs: This is where you define what kind of classification you want to run. It is an array of jobs. Each job must be surrounded by braces ({}), and each job should be separated by a comma (,). Each job must contain the following:

- method: The method of classification you want to run. There are three options:
    - MULTIVARIATE_CLASSIFICATION: Allows you to classify using a multivariate classifier.
    - UNIVARIATE_TRANSFORMATION: Allows you to classify using a univariate classifier by transforming the dataset into a univariate time series.
    - COLUMN_ENSEMBLE: Allows you to classify using univariate classifiers on specified columns of your data.
- classifier: The classifier you want to use for this job. UNIVARIATE_TRANSFORMATION and COLUMN_ENSEMBLE have access to univariate classifiers only whereas MULTIVARIATE_CLASSIFICATION has access to multivariate classifiers only. Here are the names of the available classifiers:
    - Univariate:
        - TSF_CLF: Time Series Forest

        - KNN_CLF: K Nearest Neighbors
        - PF_CLF: Proximity Forest
        - RISE_CLF: Random Interval Spectral Forest
        - EE_CLF: Elastic Ensemble
        - BOSSE_CLF: Bag of Symbols SFA Ensemblee
        - TDE_CLF: Temporal Dictionary Ensemble
        - W_CLF: Word ExtrAction for time SEries cLassification
    - Multivariate:
        - MrSEQL: Mr-SEQL classifier
        - MUSE: Multivariate WEASEL+MUSE classifer

Test Config File -

This config file runs on a sample dataset included in the NAML repository. It should serve as a test to ensure that NAML has been installed correctly.

```
{
 "filePath":"../scripts/data/2013e.csv",
 "loggingEnabled": true,
 "targetCol": "event",
 "percentTrain":0.5,
 "jobs": [
   {"method" : "MULTIVARIATE_CLASSIFICATION",
    "classifier" : "MUSE",
    "oversampling" : true
   },
   {"method": "UNIVARIATE_TRANSFORMATION",
    "classifier" : "TSF_CLF"
   },
   {"method": "COLUMN_ENSEMBLE",
    "ensembleInfo":[{
                 "classifier": "KNN_CLF",
                 "columnNum":1
                },
                {
                 "classifier": "TSF_CLF",
                    "columnNum":0
                   }]
        }
      ]
    }
```

Config File Used for Technical Evaluation -

This is the config file used to run the “AXCPT AXCAXWR Dataset for Participant 209” experiment (the results of which are included in the technical evaluation section).

```
{
   "filePath":"/home/ebuntel/NAML/NAML/scripts/data/allch_axcaxwr_09.csv",
   "loggingEnabled": true,
   "targetCol": "event",
   "percentTrain":0.75,
   "jobs": [
      {"method" : "MULTIVARIATE_CLASSIFICATION",
         "classifier" : "MUSE"
      },
      {"method" : "MULTIVARIATE_CLASSIFICATION",
         "classifier" : "MUSE",
         "oversampling" : true
      },
      {"method" : "MULTIVARIATE_CLASSIFICATION",
         "classifier" : "MrSEQL"
      },
      {"method" : "MULTIVARIATE_CLASSIFICATION",
         "classifier" : "MrSEQL",
         "oversampling" : true
      },
      {"method": "UNIVARIATE_TRANSFORMATION",
         "classifier" : "TSF_CLF"
      },
      {"method": "UNIVARIATE_TRANSFORMATION",
         "classifier" : "TSF_CLF",
         "oversampling" : true
      },
      {"method": "UNIVARIATE_TRANSFORMATION",
         "classifier" : "KNN_CLF"
      },
      {"method": "UNIVARIATE_TRANSFORMATION",
         "classifier" : "KNN_CLF",
         "oversampling" : true
      },
      {"method": "UNIVARIATE_TRANSFORMATION",
         "classifier" : "RISE_CLF"
      },
      {"method": "UNIVARIATE_TRANSFORMATION",
         "classifier" : "RISE_CLF",
         "oversampling" : true
```

```
    },
    {"method": "UNIVARIATE_TRANSFORMATION",
      "classifier" : "W_CLF"
    },
    {"method": "UNIVARIATE_TRANSFORMATION",
      "classifier" : "W_CLF",
      "oversampling" : true
    },
    {"method": "UNIVARIATE_TRANSFORMATION",
      "classifier" : "TDE_CLF",
      "parameters" : ["max_ensemble_size", 90]
    },
    {"method": "UNIVARIATE_TRANSFORMATION",
      "classifier" : "TDE_CLF",
      "parameters" : ["max_ensemble_size", 90],
      "oversampling" : true
    },
    {"method": "UNIVARIATE_TRANSFORMATION",
      "classifier" : "BOSSE_CLF"
    },
    {"method": "UNIVARIATE_TRANSFORMATION",
      "classifier" : "BOSSE_CLF",
      "oversampling" : true
    },
    {"method": "UNIVARIATE_TRANSFORMATION",
      "classifier" : "PF_CLF"
    },
    {"method": "UNIVARIATE_TRANSFORMATION",
      "classifier" : "PF_CLF",
      "oversampling" : true
    }
  ]
}
```