NirsAutoML: Building an automated classification platform for fNIRS data

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

_____
Fareya Ikram

Date: 12/13/2019
Project Advisors:

_____
Professor Erin Solovey, Advisor
Professor Rodica Neamtu, Advisor

Abstract

This project aims to enable the use of the sktime toolbox for the classification of fNIRS data by developing a testing infrastructure that will allow researchers to specify an fNIRS dataset and run sktime models on the specified data set. This tool is accompanied by a manual that presents information on how to use the tool and how to set up sktime. The tool and the manual were tested with potential users and recommendations were recorded for potential future improvements.

Table of Contents

List of Figures

List of Tables

Acknowledgements

First and foremost I would like to thank Professor Erin Solovey and Professor Rodica Neamtu on advising me on this project. Their expertise in the subject matter helped me scope this project to something that I could complete in two academic quarters. Thank you to Professor Solovey, who started meeting me much before the beginning of the academic year, so that we could find a project that would allow me to explore my interests as well as help members of the lab. Thank you to Professor Neamtu, who brought the idea of using the sktime API to the table, and continuously pointed me to sources that could help me with this project.

I would also like to thank the other MQP teams that were working with these advisors at the same time I was. Not only did they give me feedback on my work, but they were extremely supportive. Other members of the lab including Rachel, Ally, Gabi, and Sylvia helped me with testing my tool and guides. This project would not have been complete without their help.

Introduction

The goal of this project is to simplify and automate the classification component of the Brain Computer Interface (BCI) pipeline at the Worcester Polytechnic Institute (WPI) BCI lab. Towards this goal we built a tool that will allow researchers to use machine learning algorithms to classify and learn from functional near infrared spectroscopy (fNIRs) data that they have collected from experiments.

During BCI experiments, data is collected over a period of time from multiple sensors, resulting in multivariate time series data [1]. Due to the complex nature of the data, it is not trivial to utilize machine learning algorithms to classify it. Implementing and running a custom classifier appropriate for time series classification from the ground up is not only difficult and time consuming, but often makes it troublesome for other researchers to reproduce the results [2]. Even with the custom code documented in a research paper, it is unlikely that another researcher replicating the experiment will have the same implementation. There are, however, open-source tools, which can better support replication and that are designed to work with time series data specifically, such as sktime [2].

Sktime provides implementations of several state-of-the-art time series specific classification algorithms that are well-suited for fNIRS brain sensor data. This tool's developers have created a unified interface for machine learning with time series in order to, "reduce(s) confusion and enable(s) us to focus on providing advanced time series analysis capabilities for researchers and practitioners" [2]. Leveraging this toolkit has the potential to reduce the amount of time between initial data collection and classification results. Additionally, as the toolkit is open source, the implementation of sktime algorithms can easily be found, enabling reproducibility of fNIRS research.

However, sktime is a relatively new API with a steep learning curve. In fact, sktime is one of the first APIs to "to present a unified interface that can explicitly represent and link multiple distinct tasks" such as time series classification, pipelining, ensembling and data transformations [2]. Exploring the options and algorithms made available by sktime can be a time consuming and tedious process. Additionally, sktime is not designed to process fNIRS data directly. The toolbox requires data to be in a specific format to be processed by its functions and classifiers.

In this paper we present NIRS AutoML (NAML), a tool that provides an easy interface to create models with sktime and that logs information on these models, simplifying the process of

learning from and classifying fNIRS data. NAML reads fNIRS data from a csv, performs validation checks on the format and contents of the data, and prepares it for sktime. It also provides the ability to choose classification methods implemented in sktime, and specify parameters for the algorithms. After running the specified methods, it logs the performance of the given classification methods. NAML is accompanied by a manual to assist future users.

Background

Section 1: Brain Computer Interfaces (BCI)

Brain computer interfaces (BCI) are points of communication that allow signals from the brain to direct external activity[2]. In natural communication, such as shaking someone's hand, the brain would send a signal to the muscles, which would then complete the action of shaking the hand. With BCI, researchers are aiming to create an artificial system that bypasses the body's typical method of communication using electric, magnetic, or hemodynamic brain signals[3]. BCI directly measures brain activity associated with the user's intent and translates this activity into a signal for BCI applications.

BCI research can be broken down into three components[3]:

- Developing technologies such as Magnetic Resonance Imaging (MRI), electroencephalogram (EEG), and near infrared spectroscopy (NIRS) that can detect and record signals from the brain
- Using algorithms to decode information. This can be broken down to preprocessing brain data, feature extraction and selection, and classification.
- Developing a system that can integrate the decoded information into input for an interactive system

In this project, the focus will be on using algorithms to decode information, particularly data collected from fNIRS based experiments.

1.1 Supervised Machine Learning Application in BCI

Supervised machine learning is a subset of artificial intelligence, in which a computer is given data and its corresponding results or labels in order to learn a set of rules about the data. This set of rules then can be applied to classify more incoming data. This process differs from classical programming where the computer is given data and a set of rules in order to determine the results.

Supervised machine learning techniques are prevalent for classifying and understanding brain data, as signals received by machines such as NIRS and EEG are not easily interpretable. These signals can come from multiple nodes over a large period of time and can also be noisy[3]. Within the three components of BCI mentioned in Section 2.1, machine learning is essential to classification in step two.

Past research in brain computer interfaces has mainly focused on decoding and developing systems that can process EEG data. This is primarily due to the fact that in the past EEG machines were relatively inexpensive to other alternatives. Recently, however, fNIRs systems have emerged as another neuroimaging modality that is also relatively inexpensive, opening up opportunities for new experiments and applications [4].

1.2 fNIRS

Functional near infrared spectroscopy (fNIRS) is a non-invasive brain computer interface technology that collects hemodynamic data in order to monitor neuron activity in the cortex. This technology quantifies changes in hemoglobin intensity across the brain to determine the cognitive state of a person. The low cost and portable nature of this technology makes it ideal for experiments and applications where we have to monitor individual's cognitive states during situations such as safety simulations or monitoring patients during rehabilitation[4].

Section 2 : Sktime for Time Series Classification (TSC)

Researchers in fNIRS based BCI studies often choose to use machine learning techniques in order to classify or determine different cognitive processes. In this study, we explore the use of sktime, a machine learning API designed to address TSC problems, to classify fNIRS data [1]. Sktime has the potential to enable researchers to use off the shelf algorithms instead of implementing their own for each project that they do.

The sktime toolbox provides implementations of tools and algorithms that facilitate multivariate time series classification (MTSC)[1]. The tool documentation outlines three main methods for MTSC: concatenation, column ensemble, and shapelet transform. The *concatenation* method involves concatenating multivariate time series data into one one univariate series and then using a univariate classifier on the newly created univariate series [1]. The *column ensemble* method to perform MTSC specifies a classifier for each constituent univariate component of the multivariate time series. Essentially, the multivariate time series is treated as several separate univariate series. A user can then specify the classifier they would like to apply per univariate series. Finally, the predictions of these classifiers would be aggregated to provide a final accuracy score. This method is called the column ensemble method because a group or ensemble of classifiers are aggregated to get the final prediction[1]. Lastly, there is the *shapelet transform* classifier that processes the entire the multivariate time series to find shapelets, "time series

subsequences that are in some way representative of the class[7].” This classifier searches for shapelets within the multivariate series.

There are several univariate classifiers that may be used with the first two methods. The following table outlines a few univariate classifiers implemented within the sktime toolbox. Several of the implemented TSC algorithms are built upon the base decision tree classifier. A decision tree is an example of a supervised machine learning algorithm. It is built on a series of nodes each where each node asks a question about the data in order to classify it [9].

Classifier	**Basic Functionality**
Time Series Forest Classifier (TSf)	TSF is an interval based classifier built upon an ensemble of decision tree classifiers. Within each member of the ensemble a random set of intervals are selected and the basic summary statistics are computed. These statistics are then, “concatenated to form a new feature space.” [1] The library provides the ability to customize the TSF classifier.
Bag of Symbolic Fourier Transformation Symbols (BOSS) Ensemble Classifier	BOSS uses Fourier transformation to breakdown a time series into its constituent frequencies and classify the series based on this frequency[1]. BOSS classifiers are trained upon multiple windows or subsets of the series, resulting in the BOSS Ensemble Classifier.
Random Interval Spectral Ensemble (RISE)	RISE is an ensemble of decision tree classifiers. Random intervals are selected and transformed into the power spectrum that is frequency components of the series, and autocorrelation features[1]. Decision trees are then created based off this new feature space.

Table 1: Table of a select few sktime classifiers

Section 3: Integrating BCI with sktime

The motivation behind developing the sktime API was to create a unified interface for machine learning with time series in order to provide researchers “ advanced time series analysis” [1]. As mentioned in section one, using supervised learning for time series classification, is a key component for BCI research. This toolbox provides a “consistent and modular” interface that can be

used by BCI researchers to reproduce results from different experiments [1]. However, exploring the options and algorithms made available by sktime can be a time consuming and tedious process. Additionally, sktime is not designed to process fNIRS data directly. The toolbox requires data to be in a specific format to be processed by its functions and classifiers. In order to make sktime classifiers easily available for BCI researchers we aimed to develop NIRS AutoML (NAML), a tool that provides an easy interface to create models with sktime. The next section will provide an overview of NAML.

Tool Overview and Functionality

This project had three main deliverables:

1. Nirs AutoML (NAML): Automated Classification Platform for fNIRS data
2. Manual for setting up sktime (seen in Appendix A)
3. Manual for setting up NAML(seen in Appendix B)

This chapter will give an overview of the functionality of Nirs AutoML as well as provide a proof of concept of the tool with an fNIRS data set.

Section 1:Nirs AutoML (NAML) Functionality

NAML has four main functions: 1) to parse and reformat the data, 2) to classify the data, 3) to record the results of the classification, and 4) to assist users with getting started. There are two main scripts the tool consists of. The first program is `naml.py`, the script that will parse, classify, and record the data. The second program is `configuration_checker.py`. This program assists the user by checking their configuration file, and specified data file in order to save them time with errors.

The picture below shows the generic workflow of naml.py. The user sets up a JSON configuration that specifies a file path for the comma-separated value (csv) file, the target column that the user would like to classify the data into, the percentage of training data they would like the train the model on, a list of specified classifiers, and the option to log the results.

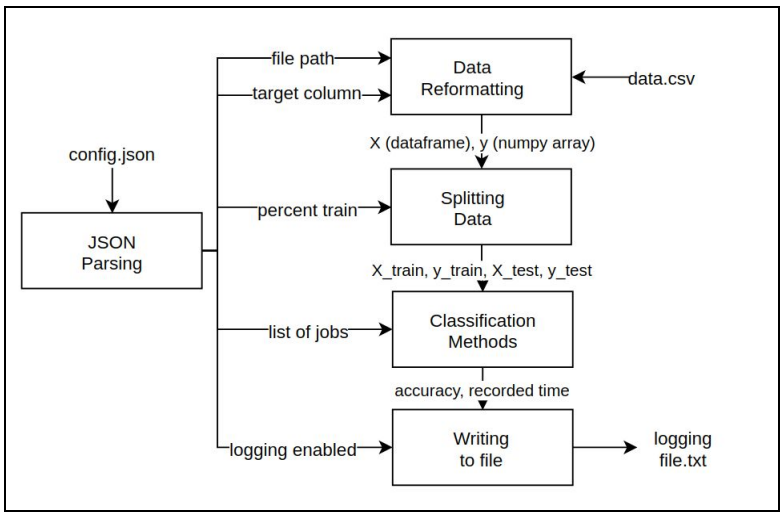


Figure 1: NAMLs workflow

1.1 Configuration File Input

In order to interact with NAML the user must create a JSON configuration formatted as shown below.

```
{
 "filePath":"../scripts/data/2013e.csv",
 "loggingEnabled": true,
 "targetCol": "event",
 "percentTrain":0.8,
 "jobs": [
    {"method": "UNIVARIATE_TRANSFORMATION",
     "classifier": "TSF_CLF"
    },
    {"method": "SHAPELET_TRANSFORM"
    }
  ]
}
```

Figure 2: Example JSON configuration file for NAML

a) `filePath` , a string, defines the path to the location of the data file you will be using.
b) `loggingEnabled,` a true or false value, determines whether the output of the run will be logged onto a text file.
c) `targetCol,` is a string that specifies which column the data will be classified into. For example, if there is a column named "event" and there are two types of events, the data will be classified in one of the two events.
d) `percentTrain,` a float value that determines the percentage of data that the models will be trained on.
e) `jobs,` a list of jobs that will run. Within each job you must specify a classification method. Depending on the method there are different parameters that can be specified. Section 1.3 will overview the different classification methods and how to specify jobs per method within a configuration file.

1.2 Data Manipulation

Sktime requires data to be formatted in a certain way before it can be used with the toolbox. NAML's data manipulation function is built to take in fNIRS data and reformat it accordingly. Consider the two images below. Figure 3 is the raw data outputted by the MATLAB graphical user interface. Several rows may represent the same event, but will contain the data collected from different sensors. Figure 4 is the output of passing this sensor data into the data manipulation

function of NAML. As you can see, the sensors' data has now been reformatted into dimensions that contain the time series per label. Each channel data is now formatted into columns: A-DC1 translates into dim_0 and A-DC2 translates into dim_0. In Figures 3 and 4, the same instance of the manipulated data is highlighted.

name	event	channel	start time	end time	1	2	3
2013e_001	100 0-Back	A-DC1	232665953.1	232695953.1	17061.4316	17151.168	16959.8223
2013e_001	100 0-Back	A-DC1	233251953.1	233281953.1	16602.2031	16672.1426	16736.8047
2013e_001	100 0-Back	A-DC1	234048359.4	234078359.4	15789.3135	15728.6113	15714.0957
2013e_001	102 2-Back	A-DC1	232876359.4	232906359.4	17371.5449	17454.6816	17400.5762
2013e_001	102 2-Back	A-DC1	233357156.3	233387156.3	16228.749	16249.8633	16261.7393
2013e_001	100 0-Back	A-DC2	232665953.1	232695953.1	237.356	239.0465	235.9601
2013e_001	100 0-Back	A-DC2	233251953.1	233281953.1	230.1286	229.4772	230.6559
2013e_001	100 0-Back	A-DC2	234048359.4	234078359.4	217.1239	216.046	215.5575
2013e_001	102 2-Back	A-DC2	232876359.4	232906359.4	242.5206	244.087	242.1639
2013e_001	102 2-Back	A-DC2	233357156.3	233387156.3	223.5448	224.1264	225.2896

Figure 3: Data outputted from MATLAB GUI

	dim_0	dim_1
0	1 17061.4316 2 17151.1680 3 16959.8223 dtype: float64	1 237.3560 2 239.0465 3 235.9601 dtype: float64
1	1 16602.2031 2 16672.1426 3 16736.8047 dtype: float64	1 230.1286 2 229.4772 3 230.6559 dtype: float64
2	1 15789.3135 2 15728.6113 3 15714.0957 dtype: float64	1 217.1239 2 216.0460 3 215.5575 dtype: float64
3	1 17371.5449 2 17454.6816 3 17400.5762 dtype: float64	1 242.5206 2 244.0870 3 242.1639 dtype: float64
4	1 16228.7490 2 16249.8633 3 16261.7393 dtype: float64	1 223.5448 2 224.1264 3 225.2896 dtype: float64

Figure 4: Data reformatted for sktime

1.3 Classification Methods

The tool currently supports four different classifiers over three different methods.

Concatenation Method

The concatenation method converts a multivariate time series into a univariate series by concatenating the series together. After this, a classifier built for univariate data, chosen by

the user, is run. In this method, we support three distinct classifiers: Time Series Forest Classifier (TSF_CLF), KNeighbors Classifier, and Proximity Forest Classifier. The following method shows three jobs each using the concatenation method.

```
"jobs": [
  {"method": "UNIVARIATE_TRANSFORMATION",
   "classifier": "TSF_CLF"
  },
  {"method": "UNIVARIATE_TRANSFORMATION",
   "classifier": "KNN_CLF"
  },
  {"method": "UNIVARIATE_TRANSFORMATION",
   "classifier": "PF_CLF"
  }
]
```

Figure 5: Example JSON for Concatenation Method

Multivariate Method

The multivariate method currently supports one classifier that supports multivariate time series: the Shapelet Transform Classifier. Here is an example of how to configure this method.

```
"jobs": [
  {"method": "SHAPELET_TRANSFORM"
  }
]
```

Figure 6: Example JSON for Multivariate Method

Column Ensemble Method

The column ensemble method consists of specifying a classifier per column or dimension of the data. The user can choose as many dimensions of data as they would like to create the ensemble. For example, the following image shows that column zero and column one will be trained using the time series forest classifier. Any classifier mentioned under concatenation method section, is also available for the column ensemble method.

```
"jobs": [
   {"method": "COLUMN_ENSEMBLE",
    "ensembleInfo":[{
                      "classifier": "TSF_CLF",
                      "parameters":["n_estimators",15,"min_samples_split",3, "min_samples_leaf",2],
                      "columnNum":1
                     },
                     {
                      "classifier": "TSF_CLF",
                      "parameters":["n_estimators",15,"min_samples_split",3, "min_samples_leaf",2],
                      "columnNum":0
                     }]
   }
 ]
```

Figure 7: Example JSON for Column Ensemble Method

Different methods can be called together in one configuration file, and a user may choose to specify as many jobs as they would like. Additionally, there is the option to parametrize the classifier based on the options provided by sktime API documentation. This can be seen in Figure 7 above as well. For any method mentioned a list by the name of parameters must be specified in the order of parameter name, parameter value, parameter name, and so on as can be seen in the image below.

```
"parameters":["n_estimators",15,"min_samples_split",3, "min_samples_leaf",2]
```

Figure 8: Example parameter input

1.4 Logging Output

If chosen, the results of the classifier may be reported. The logging output will be saved under the format date:::time.txt. Within the text file we first list the json, the csv, and the job. We then record the accuracy and the time taken to run a single job. Accuracy is the percent classified correctly, and the time is the number of seconds it took to run the model. The image below shows the time and accuracy for two different jobs to run.

```
../scripts/configFiles/config_ex4.json
../scripts/data/2013e.csv

Job: {'method': 'UNIVARIATE_TRANSFORMATION', 'classifier': 'TSF_CLF'}
Accuracy : 0.6818181818181818
Total Time : 6.0487425327301025

Job: {'method': 'SHAPELET_TRANSFORM'}
Accuracy : 0.3333333333333333
Total Time : 31.93380069732666
```

Figure 9: Example output log file

1.5 Configuration Checker

configuration_checker.py is a file that will allow users to check if the format of the configuration file is correct, if any of data for a specific parameter in incorrect, and if the specified csv file has the correct format. The following is a list of checks covered by this program.

- Parameters set in the JSON are valid parameters as described in sections 1.1 and 1.3
- fNIRS data file headers contains the following values: “name” “event”, “start time”, “end time”, “channel”

Section 2: Proof of Concept

Running sktime classification task on dual task driving study

Data Overview

The Dual Task Car Driving Study served as a test case to facilitate the development of the modular testing infrastructure. This specific study aimed to determine if periods of high workload can be distinguished from periods of low workload, based on fNIRS data. In order to do this, thirty participants were put in a driving simulation while they had to perform an auditory vocal working memory tasks, specifically n-back tests, while driving in the simulator. These tasks required users to perform a blank-back, 0-back, 1-back, and 2-back test, each test requiring about 30 seconds to complete[5]. The researchers also collected a reference signal when the participant was performing no secondary task at all. Each of these from baseline to the 2-back test, are increasing in difficulty, resulting in a higher cognitive workload.

All in all, the study suggested that participants performed worse in the tasks as difficulty increased. Additionally, preliminary results showed that the minimal value of deoxygenated hemoglobin concentration over the task period, was a good indicator of task level with significant differences found in certain sensors[5].

The experimental fNIRS data from this study is labeled for each participant. Within the data collected for each participant, there are makers for the time period in which the participant was performing a specific task (reference, blank-back, 0-back, 1-back,2-back). In total, there were 16 fNIRS sensors placed on the participant. The sensor data is marked by the signal magnitude every millisecond.

To extract this data we used a MATLAB GUI that visualizes and exports experiment data as seen below.

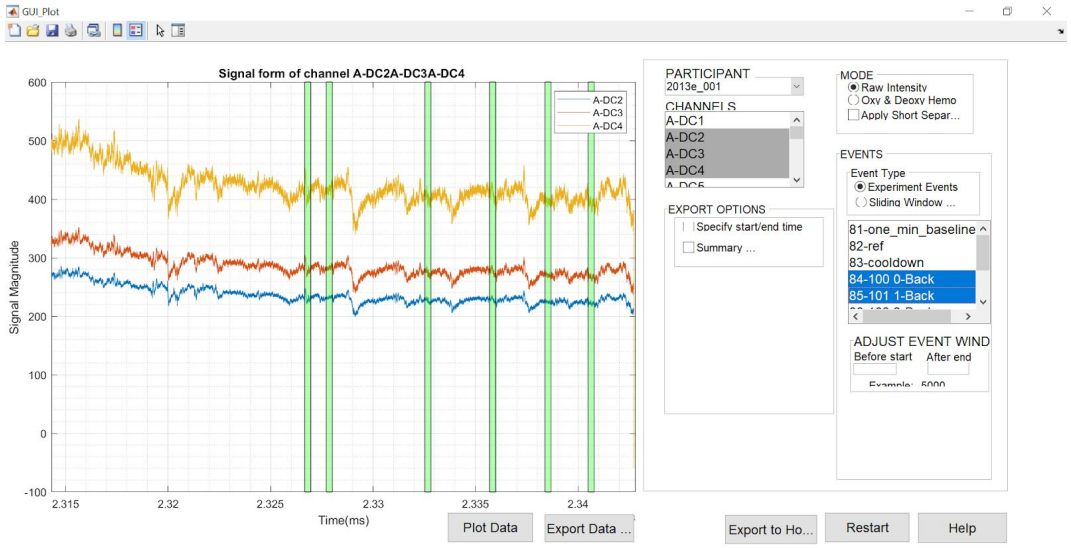


Figure 10: Matlab Graphical User Interface for fNIRS data

In the user interface shown above in Figure X, we can see data for participant "2013e_001" as selected under "PARTICIPANT." This data includes signals from three sensors, selected under the label "CHANNELS". For the Dual Task Driving Data, there are sixteen channels to choose from. The green intervals on the GUI indicate the time in which the participant was performing a given task. In this GUI, tasks are specified under "EVENTS". As the user of this interface selects a task, the time period in which that task took place is highlighted. In addition to this, the MATLAB graphical user interface shown in Figure X allows us to pick and choose the participant whose data we would like to extract and also provides simple extraction for the required data in the form of a csv file. The first few rows of the csv file looks like the following.

name	event	channel	start time	end time	1	2	3	4	5	6
2013e_001	100 0-Back	A-DC1	232665953.1	232695953.1	17061.4316	17151.168	16959.8223	16950.584	16998.0898	17038.998
2013e_001	100 0-Back	A-DC1	233251953.1	233281953.1	16602.2031	16672.1426	16736.8047	16581.0898	16521.7051	16525.6641
2013e_001	100 0-Back	A-DC1	234048359.4	234078359.4	15789.3135	15728.6113	15714.0957	15725.9717	15752.3643	15820.9854
2013e_001	102 2-Back	A-DC1	232876359.4	232906359.4	17371.5449	17454.6816	17400.5762	17273.8926	17302.9238	17321.3984
2013e_001	102 2-Back	A-DC1	233357156.3	233387156.3	16228.749	16249.8633	16261.7393	16315.8447	16387.1035	16463.6426

Figure 11: MATLAB GUI csv output

Within this dataset we extracted all the data for two "events", the 0-back and the 2-back. The 0-back task has been validated to provide a lower workload demand than the 2-back task. In the next section, we will run classification algorithms on the data to determine if they can properly

classify the data into one of these two events, and demonstrate that we can distinguish between high and low workload, based only on fNIRS data.

Setting Parameters

Once the fNIRS data file is generated as a csv, we can begin using NAML. The first step is to set up the configuration file. We will first specify the location of filePath to the location of the csv (line 1). Let's say that we would like to run each classification method (line 5). We would like to train with 50% of the data (line 4), and want to record the output to the file (line 2). Lastly, we would like to classify the data into their distinct events (line 3). In this case, an event would be the n-back test (0-back or 2-back) the user was performing at the time the data was being collected. The following configuration file would meet these requirements.

```
          {
line 1:     "filePath":"../scripts/data/2013e.csv",
line 2:     "loggingEnabled": true,
line 3:     "targetCol": "event",
line 4:     "percentTrain":0.5,
line 5:     "jobs": [
                {"method": "SHAPELET_TRANSFORM"
                },
                {"method": "UNIVARIATE_TRANSFORMATION",
                 "classifier": "TSF_CLF"
                },
                {"method": "COLUMN_ENSEMBLE",
                 "ensembleInfo":[{
                                  "classifier": "TSF_CLF",
                                   "columnNum":1
                                 },
                                 {
                                  "classifier": "RISE_CLF",
                                   "columnNum":0
                                  }]
                }
               ]
    }
```

Figure 12: Sample JSON configuration file

Reviewing Results

The following is the output of the program.

```
../scripts/configFiles/config_ex7.json
../scripts/data/2013e.csv

Job: {'method': 'SHAPELET_TRANSFORM'}
Accuracy : 50.0%
Total Time : 36.93 seconds

Job: {'method': 'UNIVARIATE_TRANSFORMATION', 'classifier': 'TSF_CLF'}
Accuracy : 53.12%
Total Time : 7.99 seconds

Job: {'method': 'COLUMN_ENSEMBLE', 'ensembleInfo': [{'classifier': 'TSF_CLF', 'columnNum': 1},
{'classifier': 'TSF_CLF', 'columnNum': 0}]}
Accuracy : 59.15%
Total Time : 2.39 seconds
```

Figure 13: Sample log output

As can be seen the accuracy of each of the models is approximately 50%,. This is essentially baseline accuracy as there are two different classes that we are classifying the data into. However, if we run the configuration file, we may notice that the shapelet transform method generally performs worse than the other two methods. In the next step of this experiment, we may choose different classifiers, classification methods, or adjust parameters for the classification methods listed above. In order to determine the best way to move forward, one would have to be a subject matter expert on the data, as well as the experiment. It is also possible to use hyper parameter tuning to determine the best steps forward. In section x.x, I outline possibilities of moving forward and creating better models.

Section 3: Feedback on NAML

NAML was tested by asking members of the Brain Computer Interface Lab to download and run the tool on their local machine. Throughout the testing session the participants gave feedback on the current tool and possible future improvements.

The users agreed that the tool is intuitive, well commented, and straightforward to use. However, certain users suggested that building a user interface for a tool would make it easier to use. Other users enjoyed the ability to view the implementation of the tool and to have the ability to modify the code as they see fit. Most users did state that writing a configuration file in the form of a JSON can be tedious and suggested that automating the generation of these configuration files can ease the workflow.

Other suggestions included the following:

- Combining the troubleshooting script and the main script (naml.py) into one master script

- Storing all logged files in one local directory
- Developing a flask API that can communicate with a web based front end
- Cache the reformatted version of the data file locally to reduce the time it takes to run the program

These suggestions will be further outlined in the Future Works section.

Methodology

This chapter will give an overview of the steps taken towards developing Nirs AutoML (NAML) as presented in *Tool Overview and Functionality* chapter. It will also describe the process of creating guides for future users. The chapter will be split into the following two sections:

1) Develop a modular infrastructure for running sktime models on fNIRS data
2) Create manuals to assist future users with sktime and NAML

The following sections will provide further detail per objective.

Section 1: Develop a modular infrastructure for running sktime models on fNIRS data

The first goal of this project was to develop a tool that allows researchers to easily run and evaluate models offered by sktime. Our tool enables users to choose a fNIRs dataset, pick a classification method and specify the needed parameters. In addition, they can choose if they would like to save the accuracy and runtime for the model or models that they run.

The tool was evaluated on its ability to take in:

1) Two or more datasets
2) Run these datasets against two or more classification methods
3) Make this tool usable for other researchers in the lab, by creating a guide detailed in section 2.2.

These tasks were split into further specific tasks completed using an iterative process over the course of eight weeks. Development was split into three components:

1. Initial development of the minimal viable product
2. Troubleshooting
3. Testing

Additionally, throughout the process, tasks and features were added based on feedback from advisors and testers. Each feature added went through this process.

The first step was to **ensure that previously collected fNIRS data can be properly reformatted to be compatible with the sktime API**. In order to do this we extracted a dataset to assist with the develop a reformatting function. We then tested this functionality by sending a secondary dataset through the script. This was done throughout the course of three iterations with

increasing modularity per iteration. For example, the first iteration only allowed a single column in the data file to be the target column, whereas in the second iteration, the target classification column was decided by the user. Additionally, we tested the function's ability to handle data with varying dimensions and sizes.

The second core goal was to **implement three distinct classification methods made from the sktime API.** We first constructed a proof of concept for each classification method, to ensure that the reformatted data is compatible with different each classification job. This step also allowed us to understand the steps required to automate each of these methods. Once these methods were automated, they were tested with the two datasets.

Additionally, we **designed a configuration file that allows the user to specify the path to the location of the dataset, the classification method they would like to use, and whether or not they want to benchmark the models for accuracy and time**. This step also required the script to properly read and parse the parameters within the configuration file. In order to test this we created several configuration files that called different classification methods with different settings.

Given feedback, we also added additional **error handling** to the tool. This was done by creating an additional script solely aimed to test the configuration file and the specified fNIRS data file. This error handling script checked the configuration file format and the format of the user specified data. The goal is that this would assist users with troubleshooting, if they get any errors.

Outcomes: With the completion of these tasks, we had a minimal viable product that was able to process two different fNIRS datasets with more than two different classification methods, achieving our goal for functionality and flexibility.

Finally, we reached out to members of the lab and guided them through the tool asking them about (1) the intuitiveness of the written guide discussed in section 2.2, (2) the flow and understandability of the tool, and (3) improvements and possible future work. These responses were recorded and documented to allow developers to continue improving on this tool.

Section 2: Create manuals to assist users with sktime and NAML

The second goal of this project was to create manuals that will enable researchers to install and run sktime on their machine. In addition to this, we aimed to develop a guideline that will allow users to run the NIRS AutoML tool on their computer.

2.1 Create a manual for “Getting started with sktime”

The aim is that this guide is to provide substantial information so that the user will be able to use sktime for classification of an online dataset with minimal effort. This guide includes the following:

- Information on the functionality provided by the sktime library
- An overview of the environment requirements for using sktime. This will include information on how to find and download needed libraries.
- Guideline on uploading and formatting datasets in order to make them sktime compatible
- Basic information and overview on the steps for setting up and running different models provided by sktime

There are four main steps to developing this guide. First, I installed sktime and ran the library on my computer. This involved configuring a development environment that will run sktime.

Next, I found a dataset to run a classification function on and recorded the steps that I took as a tutorial. In order to do this, I found time series data that is publicly available and configured sktime models to run with a chosen dataset. These first two steps involved taking notes on the process of installing and running the tutorial.

These notes were then compiled into a getting started guide that contained sections on the points bulleted above. This included organizing the notes into their distinct sections, and linking external resources where they may be helpful.

Finally, potential users of this guide to were asked to follow the outlined steps. The guide was evaluated based on the users’ ability to install sktime on their computer and run through a basic sktime tutorial. In addition, the users were asked if the guide provides substantial amount of resources to look at for further understanding on any aspect of the guide. The user feedback was used to improve the guide.

Outcomes: With the assistance of user feedback we were able to develop a comprehensive guide that will allow individuals to (1) set up sktime on their machine and (2) gain an understanding about sktime’s functionality and capability. This guide can be found in Appendix A.

2.2 Create a manual for “Getting started with NAML”

The second component of the getting started guide is to inform potential users on the functionality of the test suite developed, as outlined in Section 1. This component of the guide includes:

- A detailed overview on the functionality of this test suite
- Instructions on how to download and run the program
- Access to a sample dataset and configuration file

To complete this component of the guide, we have created a sample dataset and configuration file. This guide and the program were then tested with members of the lab for its functionality and usability. The users followed the guide which required them to download the tool on their machine and asked them to run the tool. The users gave feedback on three main areas:

1. Tool’s usability
2. Possible improvements and future work for the tool
3. Improvements on the user guide

Outcomes: This feedback for the tools usability and possible improvements was then recorded and documented as possible future work. The guide for getting started with NAML was edited and improved based on user feedback. This can be found in Appendix B. The next section will overview future work.

Future Work

In this project, we have created a tool that allows users to run multiple sktime models. However, this tool is only the beginning of exploring how useful sktime can be in solving the classification problems in the field of brain computer interfaces. This project can serve as a means for finding the best model using hyper parameter tuning. The configuration file provides users the ability to specify the parameters for each classifier as specified by the sktime API. The process of finding the most optimal model and optimal parameters can also be automated.

The tool itself can improve certain functions. For example, reformatting generally takes longer to run than individual classifiers, especially with large data files. A possible functionality may be to save the reformatted file if it is to be used in different configuration files. This is especially important as the size of the data increases. Similarly, it is also important to consider running multiple classifiers in parallel using multithreading as opposed to running them one after another, as it is done in the current implementation of the tool.

The natural next step is to build a user interface for this tool that members of the lab can use. A user interface can be implemented in many ways. On way would be to ask users a series of questions on the command line to generate a JSON configuration file. We could also create local application using one of many python graphical user interface modules such as tkinter. However, the most popular option is to create a web application so that the tool can be used from any device.

Creating a web application with this tool would require the use of an API. An API allows the frontend of the application to communicate with the backend of the application. As several web applications are developed using javascript, API often receive information as JSON. Future developers may choose to format the JSON sent from the front end of the web application in the same way the local JSON configuration files are formatted. One option to develop a python API would be to use Flask, as other projects in the Brain Computer Interface lab are creating applications using this as well.

In addition to creating a web based user interface, there is also the option of extending the back end capability of the tool by integration sktime-dl into this tool as well. Sktime-dl is a deep learning extension for the sktime library developed by the same researchers. This extension uses deep learning algorithms provided by keras, a popular deep learning library. Deep learning is gaining popularity in the field of machine learning, and the algorithms provided by this extension may prove to be helpful in solving classification problem in brain computer interface settings.

Works Cited

[1]Markus et al. 2019. sktime: A Unified Interface for Machine Learning with Time Series. (September 2019).

[2]Thibault Gateau, Gautier Durantin, Francois Lancelot, Sebastien Scannella, and Frederic Dehais. 2015. Real-Time State Estimation in a Flight Simulator Using fNIRS. *Plos One* 10, 3 (2015). DOI:http://dx.doi.org/10.1371/journal.pone.0121279

[3]Naseer N and Hong K-S (2015) fNIRS-based brain-computer interfaces: a review. *Front. Hum. Neurosci*. 9:3. doi: 10.3389/fnhum.2015.00003

[4]Anon. Functional Near-Infrared Spectroscopy (fNIRS) Cognitive Brain Monitor. Retrieved October 9, 2019 from https://technology.nasa.gov/patent/LEW-TOPS-84

[5] R. Holtzer, J.R. Mahoney, M. Izzetoglu, K. Izzetoglu, B. Onaral, and J. Verghese. 2011. fNIRS Study of Walking and Walking While Talking in Young and Old Individuals. *The Journals of Gerontology Series A: Biological Sciences and Medical Sciences* 66A, 8 (2011), 879–887. DOI:http://dx.doi.org/10.1093/gerona/glr068

[6] D. Belyusar, B. Reimer, B. Mehler, D. Afergan, J.F. Coughlin, E.T. Solovey, *Utilizing functional near-infrared spectroscopy to identify cognitive processes contributing to workload in a dual-task environment*, Society for Neuroscience Annual Meeting (Poster Presentation), Washington, D.C., November 18, 2014.

[7]Ye, L., & Keogh, E. (2009, June). Time series shapelets: a new primitive for data mining. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 947-956). ACM.

[8]Huberty, C. J., & Morris, J. D. (1992). Multivariate analysis versus multiple univariate analyses. In A. E. Kazdin (Ed.), *Methodological issues & strategies in clinical research* (pp. 351-365). Washington, DC, US: American Psychological Association. DOI:http://dx.doi.org/10.1037/10109-030

[9]Géron Aurélien. 2017. *Hands-on machine learning with Scikit-Learn and TensorFlow concepts, tools, and techniques to build intelligent systems*, Sebastopol, CA: O'Reilly Media Inc.

Appendix A: Getting Started with Sktime Guide

Introduction

If you're here you are probably already familiar with sktime. However, if you just happened to stumble upon this handbook you can learn more about the toolbox on its official page, its github page, or its documentation.

I've written this guide based on the installation process on a linux and mac machine. If you have a windows please follow the steps listed under the Windows section on this page here.

Downloading and creating an sktime environment

The first component of this section outlines creating a virtual environment. However, if you already have python set up on your machine and are comfortable downloading sktime, please skip to the starred step. NOTE: sktime is only compatible with a version of python higher than 3.6.

Creating a virtual environment with anaconda

Although this isn't necessary to create a virtual environment, it is highly recommended as it will allow you to manage your python libraries and projects more efficiently. To install anaconda visit the website and download the linux installer for **python 3.** In your downloads directory, you will find a shell script named similar to the following: Anaconda3-2019.10-Linux-x86_64.sh

Run this script by typing the following on the command line:

```
bash ./Anaconda3-2019.10-Linux-x86_64.sh
```

You will be prompted to accept the license and agreement as the program starts. Next, type the following onto the command line prompt.

```
source .bashrc
```

This command essentially refreshes the .bashrc file that was modified by the previous command that you ran. Now, we are ready to create a virtual environment. Run the following command on the commandline. Feel free to replace "yourenvname" with anything you would like.

```
conda create -n yourenvname python=3.7 anaconda
```

Next activate your environment by typing the following:

```
conda activate yourenvname
```

*Now that you are in the environment you can install stime by running the following command:

```
pip install sktime
```

That's it! You have sktime on your computer. You can deactivate the environment by typing the following command.

```
conda deactivate
```

To ensure that you installed sktime within the virtual environment. You can type the following onto the command:

```
Ipython
```

Ipython is a command shell for interactive python programming. If you type "import sktime" and click enter, you will be able to check if the sktime module is properly installed. Type "quit()" to get out of the shell.

In the next section I have outlined the basics of pandas and numpy. If you are already familiar with this, you can skip to the tutorial.

Running your first sktime experiment

Now that we know a little more about pandas and numpy, we are ready to start our own mini experiment. I would highly suggest looking at the tutorial available on the sktime documentation website.

Finding a suitable dataset

There is a great time series dataset archive at timeseriesclassification.com . This archive has both univariate and multivariate datasets that you can use. Additionally, these datasets are also formatted to work seamlessly with sktime. For this mini-experiment I will use atrial fibrillation data that can be found you here. Once you download and unzip the folder, you will find that there are file types .ts and .arff. These files are supported by sktime load functions. Now that the data is loaded, we need to load the data set.

Loading the dataset

I would suggest using jupyter notebook as we step through this tutorial as it will allow you to explore your data as you get started. Activate your environment, and launch jupyter notebook by typing the following onto your terminal.

```
>conda activate environmentname

>jupyter notebook
```

Now create a new python3 notebook and name it. Follow the code and run each cell one by one. This following jupyter notebook can also be found here.

In [1]:
```
# import library
import sktime
import numpy as np
from sktime.utils.load_data import load_from_arff_to_dataframe
```

In [2]:
```
#retrieve X and ytrain from the .arff files
#this assumes that you unzipped the file in the same folder as this .ipynb file
#in the case below this .ipynb and the AtrialFibrillation folder are stored in the the folder "examples"
#this load function returns two things
    #1) Pandas Dataframe storing data
    #2) Numpy array of corresponding labels (these could be categories that you would like to classify this into)

Xtrain, ytrain = load_from_arff_to_dataframe("../examples/AtrialFibrillation/AtrialFibrillation_TRAIN.arff")
```

In [3]:
```
# view the first five data points in the dataframe
Xtrain.head()
```

Out[3]:

	dim_0	dim_1
0	0 -0.34086 1 -0.38038 2 -0.34580 3...	0 0.14820 1 0.13338 2 0.10868 3...
1	0 -0.11362 1 -0.07410 2 -0.05928 3...	0 -0.00988 1 -0.02470 2 -0.00494 3...
2	0 -0.2079 1 -0.1683 2 -0.1980 3 ...	0 -0.02632 1 -0.04606 2 -0.08554 3...
3	0 -0.11805 1 -0.08657 2 -0.09444 3...	0 0.03510 1 0.04680 2 0.06435 3...
4	0 -0.11362 1 -0.06422 2 -0.05928 3...	0 -0.04940 1 0.01482 2 0.03952 3...

In [4]:
```
# get a brief description of of the data
Xtrain.describe()
```

Out[4]:

	dim_0	dim_1
count	15	15
unique	15	15
top	0 0.07686 1 0.06588 2 0.02196 3...	0 0.13090 1 0.13090 2 0.09520 3...
freq	1	1

In [5]:
```
# this function will allow you to see the distinct classes within the np array
np.unique(ytrain)
```

Out[5]:
```
array(['n', 's', 't'], dtype='<U1')
```

In [6]:
```
# you can also view the shape of this array by doing the following
# note that the count of labels corresponds to the rows in the dataframe
ytrain.shape
```

Out[6]:
```
(15,)
```

In [7]:
```
# repeat this for testing data
Xtest, ytest = load_from_arff_to_dataframe("../examples/AtrialFibrillation/AtrialFibrillation_TEST.arff")
```

Jupyter Notebook Tutorial

Running Models

Tutorial and API documentation can be found in the documentation page. In this tutorial, we will run through the concatenation method outlined in the cells below.

```
In [8]: # the first method we can use is concatenation method
        # this concatenates the the multivariate series into a univariate series
        # then you specify a univariarte classifier

        from sktime.transformers.compose import ColumnConcatenator
        from sktime.classifiers.compose import TimeSeriesForestClassifier
        from sktime.pipeline import Pipeline

        steps_concat = [
            ('concatenate', ColumnConcatenator()),
            ('classify', TimeSeriesForestClassifier(n_estimators=100))]
        clf_concat = Pipeline(steps_concat)
        clf_concat.fit(Xtrain, ytrain)
        clf_concat.score(Xtest, ytest)

Out[8]: 0.4
```

You may also choose to replace the TimeSeriesForest classifier with any other univariate classifier as such:

```
In [9]: from sktime.classifiers.distance_based import KNeighborsTimeSeriesClassifier
        steps_concat = [
            ('concatenate', ColumnConcatenator()),
            ('classify', KNeighborsTimeSeriesClassifier(metric='dtw'))]
        clf_concat = Pipeline(steps_concat)
        clf_concat.fit(Xtrain, ytrain)
        clf_concat.score(Xtest, ytest)

Out[9]: 0.3333333333333333
```

You may notice a Pipeline() function that runs the column concatenation. This tool allows you to run other transformations on the data as well. If you are looking for more information on the functions provided the library viewing and searching the documentation and looking at the github page can help. Searching the function "KNeighborsTimeSeriesClassifier" will give you information on all the parameters that can be used with this function. If you are trying to find the implementation you can use the import statement on top of the tutorial. For example, the k-neighbors implementation can be found in the folder sktime> classifier > distance_based. as can be seen in the first line of this code.

Familiarizing yourself with pandas and numpy

Pandas and numpy are essential when doing machine learning with python and are core building blocks when using sktime. There are endless resources when it comes to these two libraries that may come in handy. I personally recommend downloading copies Pandas Cheat Sheet and Numpy Cheat Sheet, as they will be invaluable resources when manipulating data for sktime. In sktime, the dataset excluding the labels is stored as a pandas dataframe and the labels are stored as numpy arrays. I would suggest opening up the IPython console by typing IPyhton on your terminal to follow along some of the examples. Once the console is open, type the following lines of code:

```
import pandas as pd

import numpy as np
```

Here is a brief introduction to each:

Pandas

The pandas library is popular for data manipulation and analysis. Particularly, the DataFrame objects provided by the library allows easy indexing of complicated datasets. There are a few functionalities of pandas that are a must know.

Here are two ways to create a dataset:

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(
          {"a" : [4 ,5, 6],
           "b" : [7, 8, 9],
           "c" : [10, 11, 12]},
         index = [1, 2, 3])
```

Specify values for each column.

```
df = pd.DataFrame(
     [[4, 7, 10],
      [5, 8, 11],
      [6, 9, 12]],
     index=[1, 2, 3],
     columns=['a', 'b', 'c'])
```

Specify values for each row.

To gain descriptive understanding of the dataset, here are a few functions that you may choose to use the following functions:

`df['w'].value_counts()`
Count number of rows with each unique value of variable

`len(df)`
# of rows in DataFrame.

`df['w'].nunique()`
# of distinct values in a column.

`df.describe()`
Basic descriptive statistics for each column (or GroupBy)

pandas provides a large set of **summary functions** that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

`sum()`
Sum values of each object.

`count()`
Count non-NA/null values of each object.

`median()`
Median value of each object.

`quantile([0.25,0.75])`
Quantiles of each object.

`apply(function)`
Apply function to each object.

`min()`
Minimum value in each object.

`max()`
Maximum value in each object.

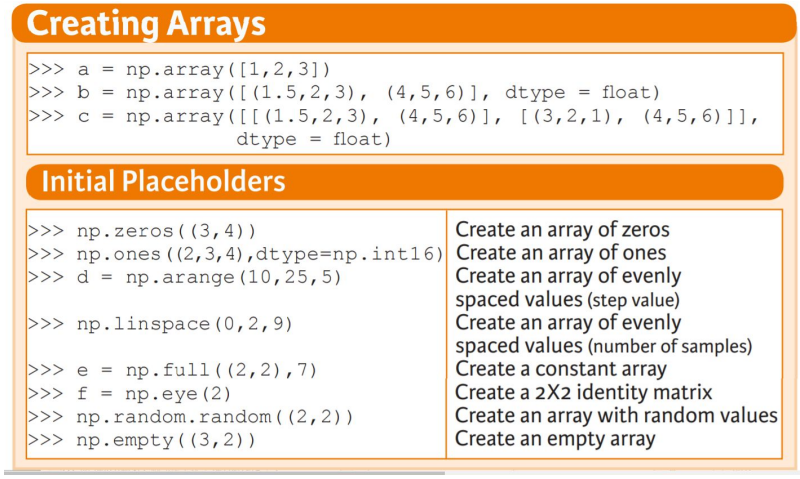
`mean()`
Mean value of each object.

`var()`
Variance of each object.

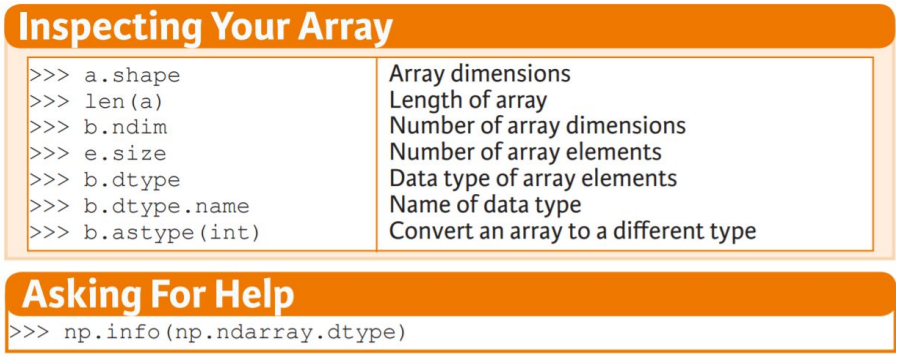
`std()`
Standard deviation of each object.

Numpy

Numpy, short for numerical python, is an open source library which provides fast mathematical computation on arrays and matrices. NumPy provides the essential multi-dimensional array-oriented computing functionalities designed for high-level mathematical functions and scientific computation.
Here are a few ways to create numpy arrays:



Here is how to gain some descriptive understanding of the data in a numpy array:



Appendix B: Getting Started with NAML Guide

Introduction

NAML is an automated classification tool built to process and classify fNIRS data collected in the Brain Computer Interface lab at WPI. The tool is built upon the sktime API, using the classifiers that are made available by the toolbox. It will allow researchers in the lab to choose classifiers, adjust parameters, and, hopefully, help them determine the best classifier for their data.

Case study: NAML user workflow for 2013 driving data

In this section we will outline the workflow of using NAML with a real fNIRS dataset collected from a distracted driving experiment aimed at determining low and high memory workload. The experimental data is exported into a csv. The following specifies the steps needed to run and use the tool effectively.

1) *Set up an environment for the tool to run*
   In order to run NAML on your computer you will need:
   a) **linux** or **mac** operating system
   b) **python 3**
   c) Installed **sktime** library
   If you want to further gain further information on installing sktime or setting up sktime you are welcome to use the getting started with sktime guide.

2) *Download the repository*
   The next step is to download the repository from the following GitHub link:
   https://github.com/erins/NAML
   The following picture shows the GitHub repository. Click the green download button on the upper right hand corner.

   The downloaded repository has two main folders. The "examples" folder contains sktime tutorials and the "scripts". The scripts file on this computer contains four main items needed to run the tool
   1. `naml.py`
   2. `configuration_checker.py`
   3. `data` folder
   4. `configurationFile` folder

   Using the terminal navigate to script folder within the downloaded folder and type the following commands that will make the python files executable :

   ```
   chmod a+x naml.py
   ```

```
chmod a+x configuration_checker.py
```

3) *Explore the data in the csv file*
   Under the data folder you will find a csv file labeled 2013e.csv. The data has multiple columns including: name, event, channel, start time, end time
   - name represents the participant. There about thirty participants in this dataset.
   - event is the type of event the participant is performing. For this dataset we have two event, 0 back and 2 back tests. This data could be indicative of high and low memory workload, respectively.
   - channel represents the sensor the data is being received from. This dataset has sixteen different sensors.
   - start time and end time are the start and end time for a particular event done by a participant
   - The remaining columns represent the data points collected by the machine over time at the sampling frequency

   Several columns represent the same event but with different sensors. One way to distinguish between the different events is to look at the start time or end time

4) *Set up the configuration file*
   The following picture shows an example configuration file that can be found under `configFiles`.

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

   a) filePath , a string, defines the path to the data you will be using
   b) loggingEnabled, true or false, determines whether the output of the run will be logged onto a text file
   c) targetCol is a string that specifies which column the data will be classified into

d) percentTrain is a float value that determines the percent of data that will be trained on
e) jobs is a list of jobs that will be run. Within each job you must specify a method. If the type of method is univariate transformation, you can also specify a classifier.

5) *Check correctness of the configuration file*
On the command line type the following command:
./configuration_checker.py ../scripts/configFiles/example_config.json
This script is aimed at helping you detect any errors with configuration file.

6) Run the classification jobs on the data
On the command line type the following command:
./naml.py ../scripts/configFiles/example_config.json

7) *Review logged output*
In your local directory navigate to the logs folder. In this folder you will find a .txt file that will show the percent accuracy and the time it took for the model to run.