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.

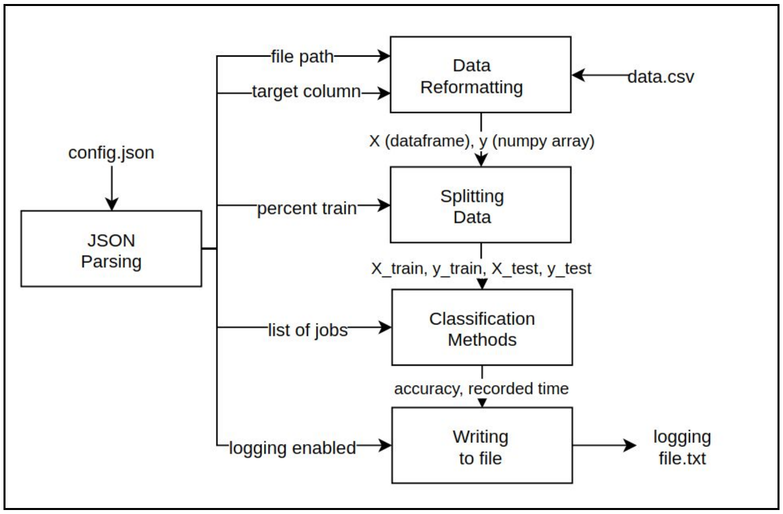
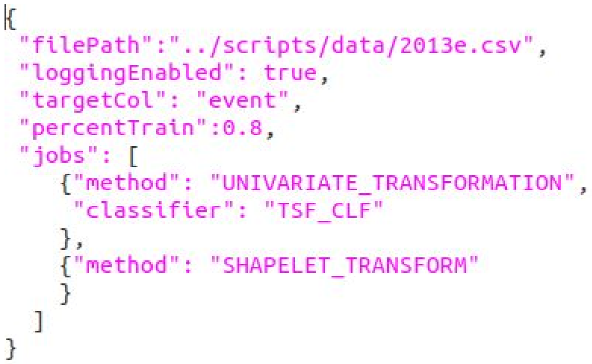


Figure 1: NAMLs workflow

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.

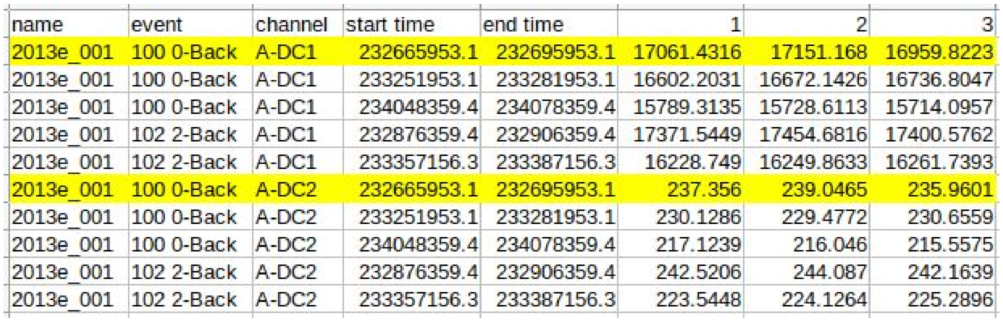
c) `targetCol,` is a string that specifies which column the data will be classified into. For example, if there is a column named "event" and there are two types of events, the data will be classified in one of the two events.

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.

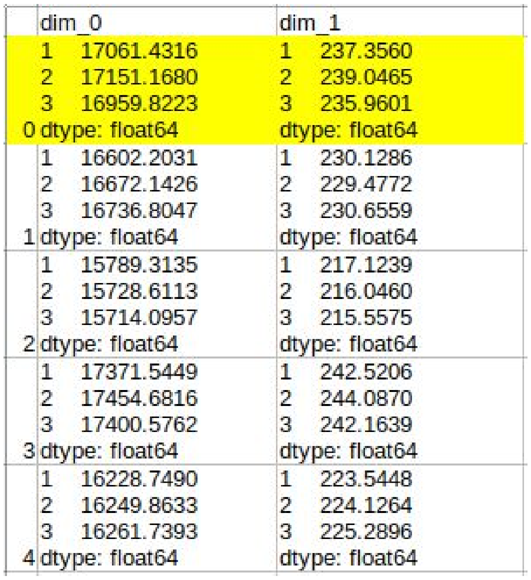
1.2 Data Manipulation

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

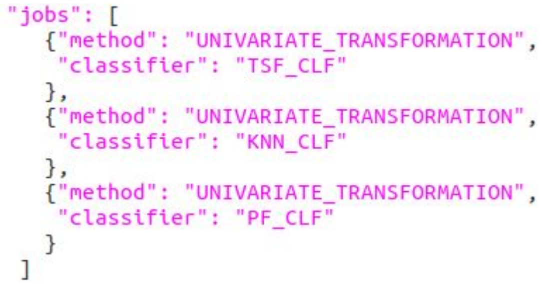
Figure 4: Data reformatted for sktime

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.

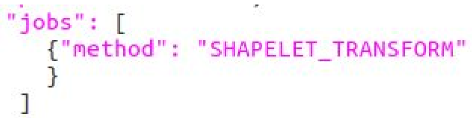


```
"jobs": [
   {"method": "UNIVARIATE_TRANSFORMATION",
    "classifier": "TSF_CLF"
   },
   {"method": "UNIVARIATE_TRANSFORMATION",
    "classifier": "KNN_CLF"
   },
   {"method": "UNIVARIATE_TRANSFORMATION",
    "classifier": "PF_CLF"
   }
 ]
```

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.

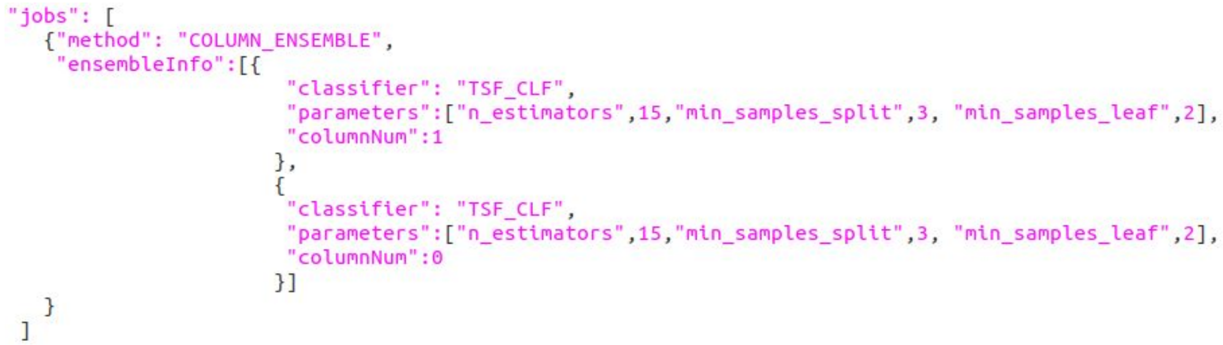


```
"jobs": [
   {"method": "SHAPELET_TRANSFORM"
   }
 ]
```

Figure 6: Example JSON for Multivariate Method

Column Ensemble Method

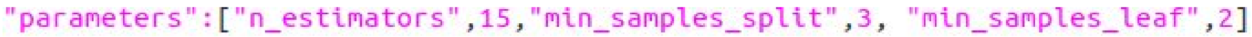
The column ensemble method consists of specifying a classifier per column or dimension of the data. The user can choose as many dimensions of data as they would like to create the ensemble. For example, the following image shows that column zero and column one will be trained using the time series forest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

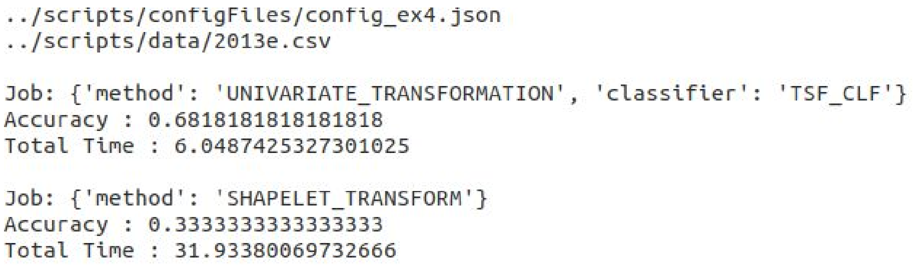


```
"parameters":["n_estimators",15,"min_samples_split",3, "min_samples_leaf",2]
```

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

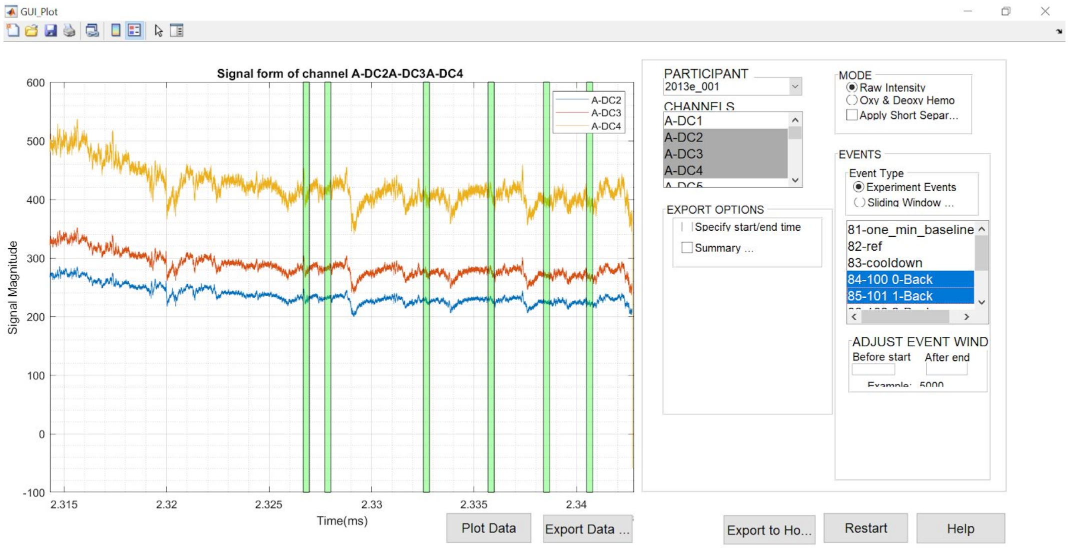
Data Overview

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.



PARTICIPANT
2013e_001

CHANNELS
A-DC1
A-DC2
A-DC3
A-DC4

MODE
● Raw Intensity
○ Oxy & Deoxy Hemo
☐ Apply Short Separ...

EXPORT OPTIONS
☐ Specify start/end time
☐ Summary ...

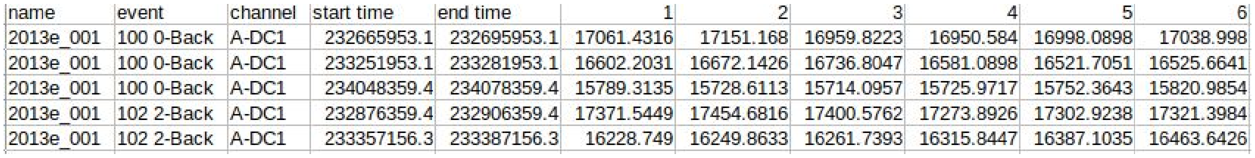
EVENTS
Event Type
● Experiment Events
○ Sliding Window ...

81-one_min_baseline
82-ref
83-cooldown
84-100 0-Back
85-101 1-Back

ADJUST EVENT WIND
Before start
After end
Example: 5000

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.



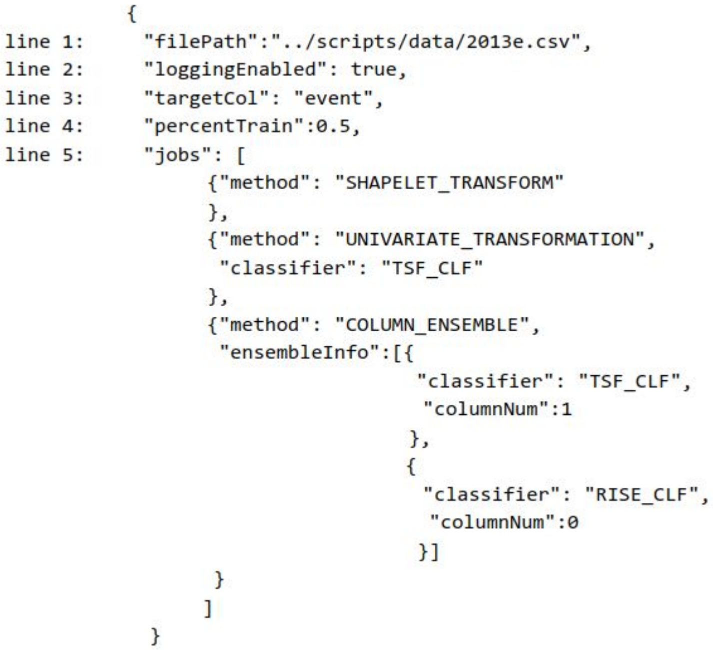
name	event	channel	start time	end time	1	2	3	4	5	6
2013e_001	100 0-Back	A-DC1	232665953.1	232695953.1	17061.4316	17151.168	16959.8223	16950.584	16998.0898	17038.998
2013e_001	100 0-Back	A-DC1	233251953.1	233281953.1	16602.2031	16672.1426	16736.8047	16581.0898	16521.7051	16525.6641
2013e_001	100 0-Back	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.

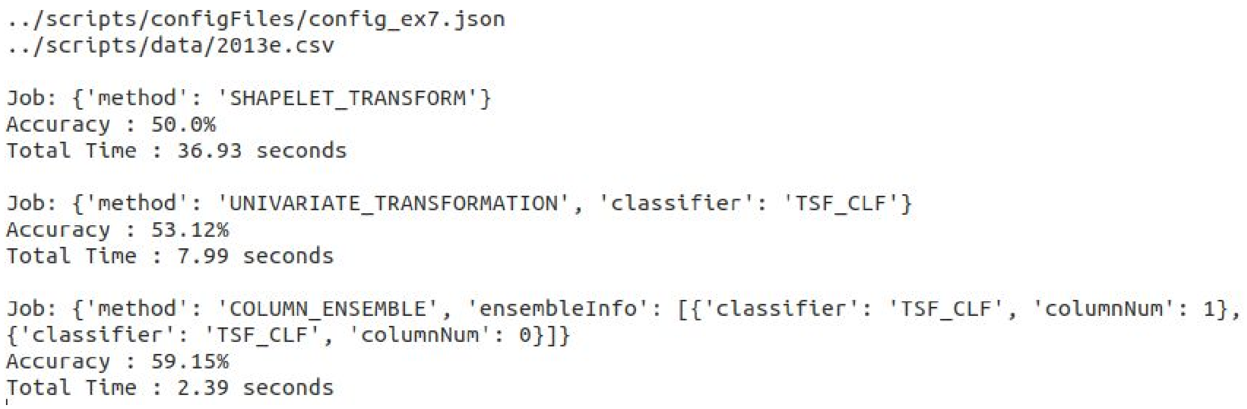


```
                {
line 1:           "filePath":"../scripts/data/2013e.csv",
line 2:           "loggingEnabled": true,
line 3:           "targetCol": "event",
line 4:           "percentTrain":0.5,
line 5:           "jobs": [
                        {"method": "SHAPELET_TRANSFORM"
                        },
                        {"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
- Connect the tool with the fNIRS data database directly