Making Brain Computer Interface Research More Accessible with Tool Tutorials

An Interactive 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:
Project Advisors:

Professor Erin Solovey, Advisor
Professor Rodica Neamtu, Advisor

Abstract

Modern Brain Computer Interaction (BCI) research commonly uses advanced data analysis tools to investigate patterns in neural data. These tools, which often use machine learning, are utilized to gain insights from recorded brain data that is too complex to be effectively examined manually. Tools like this are convenient for research use, but their complexity often makes them difficult to learn. Tutorials, therefore, are an essential companion to any useful tool. The contents of this report include a series of tutorials on the NIRS Automated Machine Learning (NAML) and BrainEx data analysis tools. These powerful tools can be used by anyone if they reference these tutorials. Some of the tutorials were created using Jupyter Notebook [2] which allows users to interact with the tutorials as they progress.

Table of Contents

Introduction

BrainEx and NAML were both developed in WPI's HCI Lab as data analysis tools for use on time series data. Time series data is data that is indexed by time. Common examples include heart rates, stock prices, and neural data. However, while both NAML and BrainEx are meant to analyze time series data, BrainEx and NAML have very different use cases.

BrainEx is a time series similarity search tool. Users input a section of their time series data into BrainEx and BrainEx returns a list of other sections of the users' data that are similar. This allows researchers to quickly and efficiently find similarities in their data. These similarities can be used to gain insight into what kinds of brain activity are similar or dissimilar. This can even be useful in differentiating between different cognitive states. BrainEx itself is a library for the Python programming language.

NAML, on the other hand, is a machine learning application for automatically running classifiers on neural data. These classifiers attempt to sort rows of neural data into one of multiple categories. For example, NAML's classifiers have been used to determine what kind of event was taking place at the time the neural data was recorded. This allows researchers to obtain insights from power machine learning algorithms without having to implement those algorithms themselves. NAML is a console application that was written in and requires Python.

Both of these tools are powerful but their implementations are complex so they can be difficult to learn to use. This report was written to help alleviate that problem by providing guides to new users of these tools. These tutorials will guide readers through the installation of BrainEx and NAML as well as their usage.

Tutorials

Jupyter Notebook

A Brief Guide to Jupyter Notebooks

Jupyter Notebook is a commonly used data-science tool that allows its users to run their Python scripts as they're writing them. Jupyter Notebooks consist of chunks of Python code that can be executed independently and each give their own output. This is very convenient for making interactive tutorials so I have written some of the tutorials in this repository as Jupyter notebooks. For those of you who are unfamiliar with Jupyter Notebook, here is a quick guide as well as some useful links.

Installation

The easiest way to install Jupyter Notebook is to install Anaconda. If you already installed Anaconda as a part of installing BrainEx then you can skip the rest of this step.

If you have installed Python without Anaconda, then use the following terminal command to install Jupyter notebook:

pip install notebook

Using Jupyter Notebooks

Jupyter Notebook has the same starting command on Linux, Windows, and Mac OSs:

jupyter notebook

The above command starts a Jupyter Notebook server. To access your server simply put the link printed when you started the server into any browser. Yours will look somewhat different but here is an example link:

- http://localhost:8888/?token=a1c2065e7cde2991015bf5900ffa0c8a4e1d95eb9fe3464e

When you are done with your server you can shut it down by terminating the process from the terminal you used to start it.

BrainEx Installation Guide

Installation Process

The installation process for BrainEx is described below.

Note: If you already have Python installed feel free to ignore Step 1.

Step 1. Python

BrainEx is a Python library so the first step to using BrainEx is to install Python. BrainEx is written in Python 3.6, but any version 3.6 or later will work.

For installing Python you have two options. Either install Anaconda, which also includes libraries that are used in other parts of these tutorials, or install Python from the Python website here.

The versions of Python found at the websites above come packaged with the pip package manager. We will use this package manager in the next step to install BrainEx's dependecies.

Step 2. Install Python Dependencies

BrainEx requires certain Python libraries to be installed before usage. As mentioned above we will use Python's package manager pip to install them.

python -m pip install numpy

python -m pip install cython

Now, navigate to the "brainex" folder inside of the BrainEx repository. This repo should have a requirements.txt file. If you run the below command in this folder, it will automatically install many of BrainEx's dependencies.

python -m pip install -r requirements.txt

Finally, run these last commands to install the remaining dependencies.

python -m pip install git+git://github.com/ApocalyVec/fastdtw.git

python -m pip install h5py

python -m pip install boto3

Step 3. Run the Setup

The main folder of the BrainEx repository should include a “setup.py” file. This file checks if you have the correct dependencies installed (which we did in Step 2), and installs the BrainEx library itself.

To run the setup.py file you just have to use the following command:

python -m pip install .

This should run the setup.py script which will install the BrainEx library on your computer.

Step 4. Check the Installation

Now that you’ve installed the BrainEx library it’s best to test if your installation is working correctly. This can be done very easily.

First, run a Python editor using the python command:

python

This should change your command line to a Python command line allowing you to write Python code in your terminal. To check if BrainEx is correctly installed we can simply attempt to import it:

import brainex

If the above command executes without error then BrainEx is correctly installed!

You should now be able to call BrainEx from any python file running on your computer. If you would like more information on how to use BrainEx I would suggest looking at the other tutorials included in this guide. In particular, the BrainEx demo will give you an overview of the system’s features.

Using Spark with BrainEx

Much of BrainEx’s efficiency comes from its use of multiprocessing in preprocessing your data set. BrainEx allows you to choose which of two parallelization systems you want BrainEx to use for this multiprocessing. You can use the parallelization system built with just Python, or the Apache Spark parallelization system. The Spark system uses the library pyspark and requires an installation of Spark on your computer. Conveniently, Spark should be downloaded already as part of one of the libraries installed in Step 2. This means, if you downloaded BrainEx’s dependencies without error, you should be able to use BrainEx’s Spark option.

If, however, you were not able to install Spark as a part of Step 2, then this guide should help.

BrainEx Demo

This is a demo for the BrainEx timeseries exploration system.

This demo is meant to show the capabilities of BrainEx, but not to be an exhaustive guide. For more information please

consult the other tutorials included in this repository.

from brainex.utils.gxe_utils import from_csv

data = './ItalyPower.csv' # This is a small timeseries dataset we're going to test BrainEx on.

Step One.

The first step to analyzing a dataset with BrainEx is to create a BrainEx Engine object.

BrainEx Engine objects are what we use to utilize most of the functions of BrainEx.

The parameters for the from_csv method will be explained in more detail in a later tutorial, but for now just know that it

allows us to create a BrainEx Engine object from a .csv data file.

brainExEngine = from_csv(data, feature_num = 0, num_worker = 2, use_spark = False, driver_mem = 4, max_result_mem = 2, _rows_to_consider=12)

msg: from_csv, feature num is 0, auto-generating uuid

Not using z-normalization

[91m Genex Engine: Using Python Native Multiprocessing[00m

Step Two.

The second step in the system is to preprocess the dataset we created a BrainEx Engine object with. We can accomplish

this with the BrainEx Engine build function. This preprocessing is the most time-consuming part of the BrainEx process

so don't be concerned if it runs for a while.

brainExEngine.build(st = 0.1)

{'self': <brainex.database.BrainexEngine.BrainexEngine object at 0x7fa2b77c1e10>, 'st': 0.1, 'dist_type': 'eu', 'loi': None, 'verbose': 1, '_group_only': False, '_use_dss': True, '_use_dynamic': False}

Having preprocessed the dataset we know want to choose a query to give to the BrainEx system. BrainEx will return

subsequences that are similar to the query.

This line just chooses a random sequence from 'ItalyPower.csv' to be our query.

queryseq = brainExEngine.get_random_seq_of_len(15, seed=1)

Step Three.

The third (and final) step in the BrainEx system is querying. When we give the BrainEx Engine's query function a

subsequence of the dataset it will find the k most similar subsequences to your query.

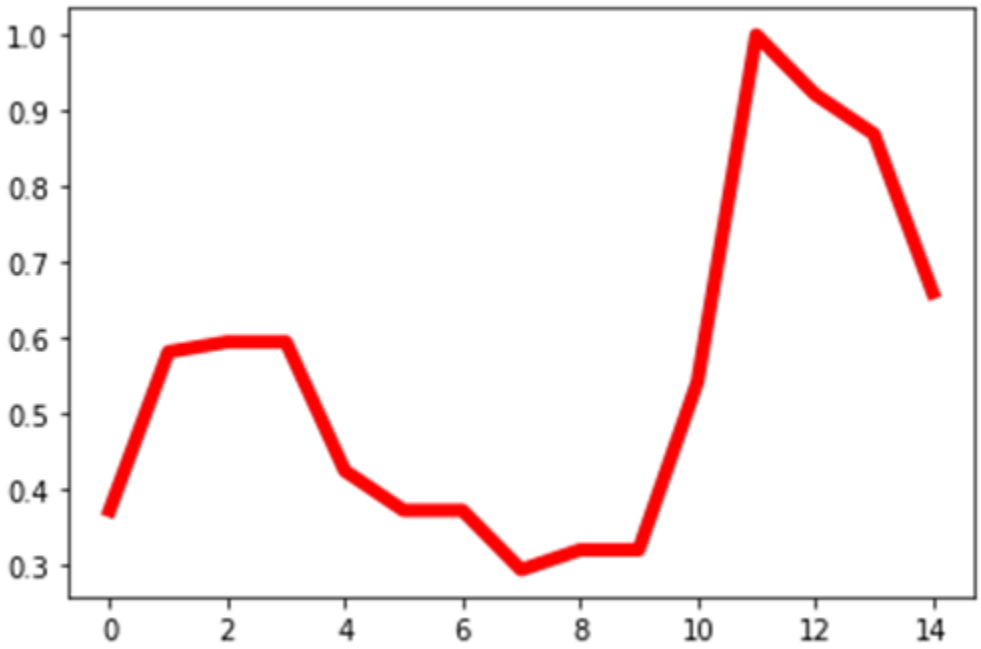
results = brainExEngine.query(query = queryseq, best_k = 5)

Now, let's visualize our query sequence using the popular Python library matplotlib.

import matplotlib.pyplot as plt

plt.plot(queryseq.fetch_data(brainExEngine.data_normalized), linewidth=5, color='red')

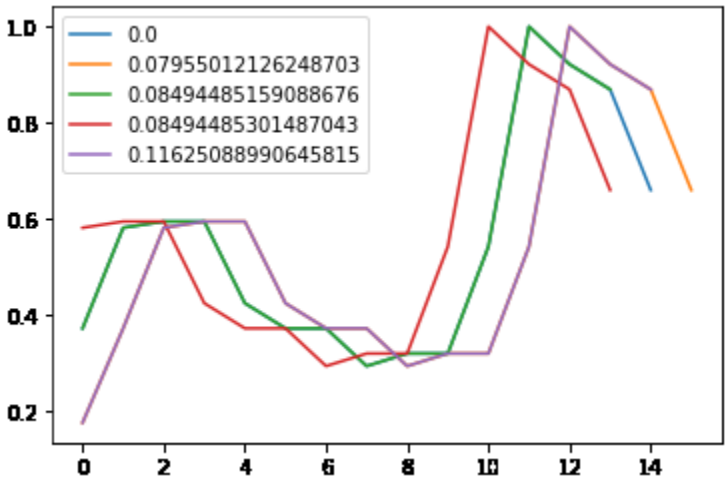
[<matplotlib.lines.Line2D at 0x7fa2b475ab90>]



Lastly, let's visualize the results BrainEx gave us for subsequences similar to our query.

(Some of the results may be overlapping. This is normal.)

```
for res in results:
        plt.plot(res[1].fetch_data(brainExEngine.data_normalized), label=str(res[0]))
plt.legend()
plt.show()
```



BrainEx Saving and Loading

This is a tutorial for BrainEx's built-in saving and loading functions.

First, let's create a BrainEx Engine and Preprocess a dataset.

```
from brainex.utils.gxe_utils import from_csv

data = './ItalyPower.csv'

brainExEngine = from_csv(data, feature_num = 0, num_worker = 2, use_spark = False,
driver_mem = 4, max_result_mem = 2, _rows_to_consider=12)
```

msg: from_csv, feature num is 0, auto-generating uuid

Not using z-normalization

[91m Genex Engine: Using Python Native Multiprocessing[00m

```
brainExEngine.build(st = 0.1)
```

{'self': <brainex.database.BrainexEngine.BrainexEngine object at 0x7f8ce5b27210>, 'st': 0.1, 'dist_type': 'eu', 'loi': None, 'verbose': 1, '_group_only': False, '_use_dss': True, '_use_dynamic': False}

Now we have a BrainEx Engine object that has already been built.

We can now save this object for later.

```
brainExEngine.save('~/savelocation') # The only parameter of the save function is the path you
want the object saved to.
```

Now let's use Python's os library to check if the engine saved.

```
import os

os.listdir('~/savelocation')
```

['subsequences.gxe',

'conf.json',

'clusters.gxe',

'cluster_meta_dict.gxe',

'data_normalized.gxe',

'build_conf.json',

'data_original.gxe']

Looks about right so let's try loading the engine we just saved.

For this we'll use BrainEx's other BrainEx engine object creation function: from_db

from brainex.utils.gxe_utils import from_db

newEngine = from_db('~/savelocation', num_worker = 2)

[91m Genex Engine: Using Python Native Multiprocessing[00m

We now have our saved, then loaded, engine ready for use in the newEngine variable. Let's try querying.

queryseq = newEngine.get_random_seq_of_len(12, seed=1)

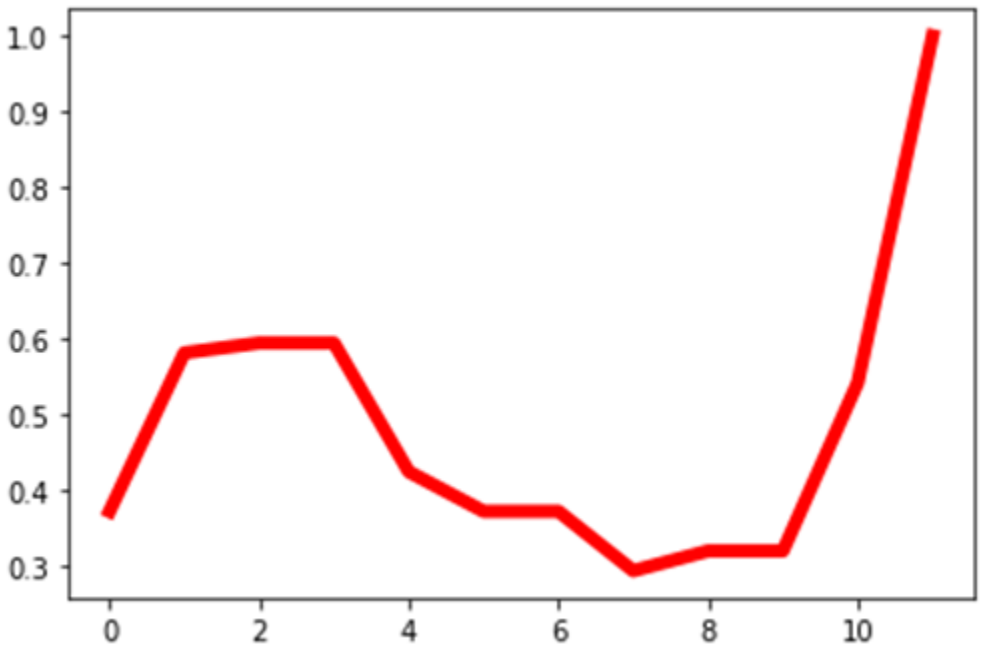
results = newEngine.query(query = queryseq, best_k = 5)

import matplotlib.pyplot as plt

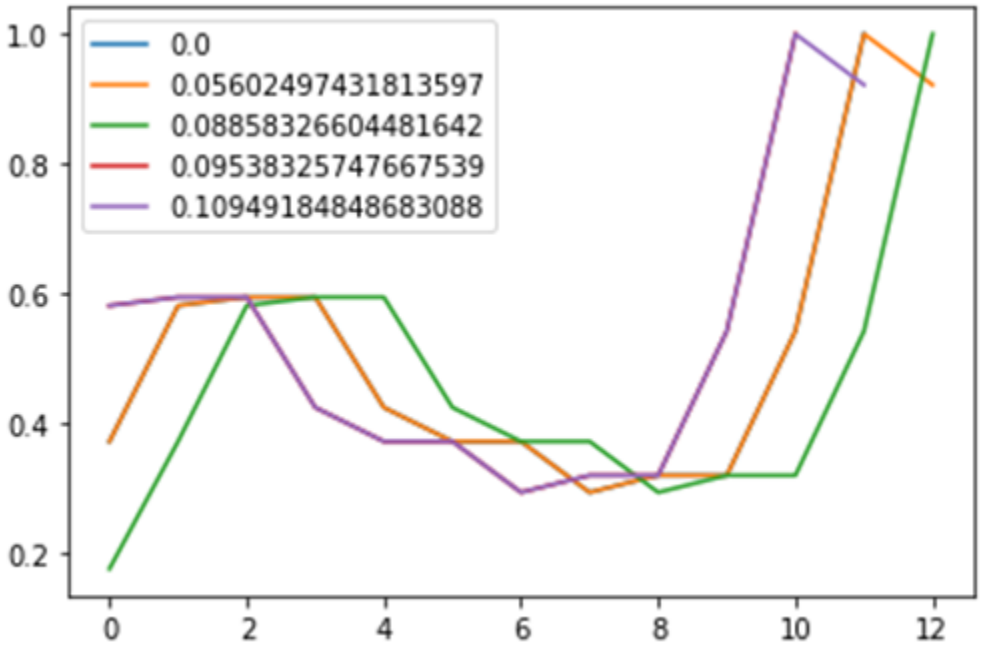
Here is the query sequence

plt.plot(queryseq.fetch_data(newEngine.data_normalized), linewidth=5, color='red')

[<matplotlib.lines.Line2D at 0x7f8cc5dcb890>]



```
# And here are the results
for res in results:
        plt.plot(res[1].fetch_data(newEngine.data_normalized), label=str(res[0]))
plt.legend()
plt.show()
```



BrainEx Parameters

This tutorial is meant as an introduction to some of BrainEx's important parameters. Knowing these parameters

will allow you greater control over the functions of BrainEx.

from brainex.utils.gxe_utils import from_csv, from_db

First, let's discuss the from_csv and from_db functions.

These functions (as you may recall from the saving + loading tutorial) are used to create a BrainEx Engine object. from_csv

creates one based off of a datafile, and from_db loads a previously saved BrainEx Engine. Both from_csv and from_db have

parameters that allow you to define the resources given to your BrainEx Engine.

from_csv

data = './ItalyPower.csv' # The first parameter of from_csv is data. This is a path to the data file you wish to use.

feature_num = 0 # Feature num is the number of columns before the timeseries data begins in your data file. Some data files

have data stored before the timeseries itself begins on each row. If your dataset has these columns then feature_num must

be equal to the number of these columns. The dataset we're testing on doesn't have any so feature_num is 0.

use_spark = False # If this bool is set to True then your BrainEx Engine will use Spark. If it's set to False it won't.

header = 0 # This is the number of header rows before the data in your dataset. Header rows usually name the different columns

and most datasets either have one or none. Our dataset this time doesn't have any.

driver_mem = 1 # This parameter defines the amount of memory (RAM) given to the main BrainEx process. In effect,

this is the number of gigabytes of RAM that BrainEx will run with. This number is limited by the amount of RAM available

on your computer. For larger datasets I would suggest using more RAM, but our test dataset is small so 1 gb should be fine.

max_result_mem = 1 # This parameter defines the amount of memory (RAM) given to make the result. As with driver_mem,

larger datasets will require more RAM. If you are unsure how much to use I would suggest giving driver_mem and max_result_mem

half of your memory each.

_rows_to_consider = None # By default BrainEx uses all rows of the inputted dataset, but if you would like to limit the number

# of rows that the system will search through then you can use this parameter. If _rows_to_consider is not included in the

from_csv invocation then the whole dataset will be searched through. If in doubt I would suggest not using this parameter.

num_worker = 1 # num_worker sets the number of cores that BrainEx will use to preprocess the dataset. The larger the number of

cores the faster your dataset will be processed, but you are limited by the number of cores in your computer. num_worker must

be at least 1.

Now, let's use the parameters we set above.

brainexengine = from_csv(data, feature_num = feature_num, use_spark = use_spark, header = header, driver_mem = driver_mem, max_result_mem = max_result_mem, num_worker = num_worker)

msg: from_csv, feature num is 0, auto-generating uuid

Not using z-normalization

[91m Genex Engine: Using Python Native Multiprocessing[00m

Now that we have a working BrainEx Engine object let's try saving it so we can use it for from_db.

brainexengine.save('~/savelocation')

from_db

path = '~/savelocation' # Path, the first parameter of from_db, points to the saved BrainEx Engine. In this case, we just saved

our engine to '~/savelocation' so we should set path to the same value.

num_worker = 1 # Like in from_csv, num_worker is the number of cores given to BrainEx. It must be at least 1, and can't be

higher than the number of cores available in your computer.

driver_mem = 2 # Again, driver_mem is the same here as in from_csv.

max_result_mem = 2 # max_result_mem is the same here as in from_csv.

bxe = from_db(path = path, num_worker = num_worker, driver_mem = driver_mem, max_result_mem = max_result_mem)

[91m Genex Engine: Using Python Native Multiprocessing[00m

Now that we've explored the functions that create BrainEx Engines, let's look at their functions of the engines themselves.

The first is the build function.

brainexengine.build

st = 0.1 # st stands for similarity threshold. The similarity threshold can be between 0 and 1 (but not 0 or 1) and it controls

how the clusters are built during preprocessing. Larger values result in larger clusters that are few in number. On the other

hand, smaller values result in more, smaller clusters. There is no one correct value, but 0.1 is often a good starting place.

dist_type = 'eu' # The distance type controls the distance metric used in clustering. Preprocessing with different distance

metrics can result in different results. The available options are 'eu' (Euclidean), 'ma' (Manhattan), and 'ch' (Chebyshev).

loi = None # loi stands for length of interest. If you are only interested in subsections of a certain length, then you can use

loi to preprocess only subsequences of that length. However,if you do so, you will only be able to query subsequences with the

same lenght as the loi. If set to None, BrainEx will preprocess subsequences of every length. I would suggest not using this

parameter unless you have a specific goal in mind.

verbose = 1 # The parameter just controls whether or not BrainEx will print information about the build process as it goes.

If set to 1 it will print this info, and if set to 0 it will not.

brainexengine.build(st = st, dist_type = dist_type, loi = loi, verbose = verbose)

{'self': <brainex.database.BrainexEngine.BrainexEngine object at 0x7f3c024b9f90>, 'st': 0.1, 'dist_type': 'eu', 'loi': None, 'verbose': 1, '_group_only': False, '_use_dss': True, '_use_dynamic': False}

Finally, let's examine the important parameters used in querying.

brainexengine.query

query = brainexengine.get_random_seq_of_len(15, seed=1) # Query is the subsequence of the dataset that you want BrainEx to match

against. In this case I am using the get_random_seq_of_len function to set the query to a random subsequence of our dataset.

best_k = 5 # best_k is used to define the number of matches you wish BrainEx to find. It must be at least one.

loi = None # loi stands for length of interest, and, like in the build function, it controls the length of the sequences that

BrainEx searches through. If you have a specific length you want to look for then you can use this parameter, but otherwise

leaving it as None causes BrainEx to search through all lengths in the dataset.

exclude_same_id = False # This bool controls whether or not the query sequence will be excluded from the results. By default it

is set to False which generally causes the query sequence to be included in the results.

overlap = 1.0 # Overlap is a float that must be between 0 and 1 (inclusive). If overlap is used, then the results returned by

the query function will overlap at most overlap%. If your results are too similar you can use this parameter to ensure that

only results that are dissimilar to each other will be included.

brainexengine.query(query = query, best_k = best_k, loi = loi, exclude_same_id = exclude_same_id, overlap = overlap)

[(0.0, <brainex.classes.Sequence.Sequence at 0x7f3c00b02d10>),

(0.0765400441427873, <brainex.classes.Sequence.Sequence at 0x7f3c00ac0ad0>),

(0.08013465954685774, <brainex.classes.Sequence.Sequence at 0x7f3c02653690>),

(0.09014727756125093, <brainex.classes.Sequence.Sequence at 0x7f3c02653250>),

(0.09055574294203451, <brainex.classes.Sequence.Sequence at 0x7f3c026532d0>)]

NAML

NAML Installation Guide

NAML Installation Tutorial

Installation Process

The installation process for NAML is described below.

Note: If you already have Python installed feel free to ignore Step 1.

Step 1. Python

BrainEx is written in Python 3.7, but almost any Python version 3.6 or later will work.

Installing Python is as simple as visiting this link, choosing the version of Python you want, and running the executable.

The versions of Python found on website above come packaged with the pip package manager. We will use this package manager in the next step to install NAML's dependecies.

Step 2. Dependencies

The NAML repository comes packaged with a requirements.txt file. This file, conveniently, can be read by Python's package manager to automatically install NAML's required libraries. To do this, simply navigate to NAML's main folder (which contains the requirements.txt file), and run the following command:

python -m pip install -r requirements.txt

This will install each dependency listed in the requirements.txt file. If they install correctly, then NAML is now ready for use.

Step 3. Checking the Installation.

After installing NAML for the first time I would recommend giving it a test run to ensure that the dependencies we installed in step 2 are working correctly. NAML comes packaged with a few pre-made config files that are well suited to this purpose. Let's try running one. Run the following command in the /scripts/ folder of the NAML repository:

python naml.py ./configFiles/example_config.json

If the above command runs without presenting any errors then NAML has been correctly installed.

NAML Config File Parameter Guide

General Parameters

The following parameters are required for every NAML config file:

- 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

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:

1. MULTIVARIATE_CLASSIFICATION: Allows you to classify using a multivariate classifier.
2. UNIVARIATE_TRANSFORMATION: Allows you to classify using a univariate classifier by transforming the dataset into a univariate time series.
3. 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:

1. TSF_CLF: Time Series Forest
2. KNN_CLF: K Nearest Neighbors

3. PF_CLF: Proximity Forest
4. RISE_CLF: Random Interval Spectral Forest
5. EE_CLF: Elastic Ensemble
6. BOSSE_CLF: Bag of Symbols SFA Ensemblee
7. TDE_CLF: Temporal Dictionary Ensemble
8. W_CLF: Word ExtrAction for time SEries cLassification

Multivariate:

1. MrSEQL: Mr-SEQL classifier
2. MUSE: Multivariate WEASEL+MUSE

Future Work

The guides in this report should allow readers to become proficient with the NAML and BrainEx tools, but there are improvements to this report that could improve the learning experience for readers.

Firstly, the tutorials included in this report are accurate at the time of writing, but NAML and BrainEx are constantly being updated. In particular, NAML is often updated to include new classification algorithms. These would need to be added to the NAML parameters tutorial. Any future modifications to these tutorials will have to update them as any such new features / options are added to NAML and BrainEx.

Another possible addition is a developer's guide to BrainEx. The guides in this report are useful from the perspective of a new user to BrainEx, but there are no guides available yet to help developers that are working to modify BrainEx. NAML has a rudimentary developers' guide included in the NAML 2.0 report [3], but BrainEx has no such guide. The creation of a developers' guide to BrainEx could make further modification of the system easier to accomplish.

Finally, though these guides should work on any operating system and computer setup, NAML and BrainEx tend to work better when run on Linux. This is often problematic as many researchers in the HCI Lab use Windows as their primary operating system. For this reason I think that a guide on setting up the Windows Subsystem for Linux (WSL) would be a helpful addition. Such a guide would help standardize the operating systems of researchers in the lab which would be beneficial in the long run.

Conclusion

The high complexity of brain computer interface research makes the onboarding process for new researchers very difficult. Many tools used by BCI researchers require significant time investments to learn as they employ very advanced data analysis methods. Their tutorials, too, are often obtuse. This is particularly true for the researchers who most need to use them - those who don't have experience in data analysis. It is for these reasons that thoroughly written tutorials are so critical for BCI data analysis tools. With the use of Jupyter Notebook we can now also make truly interactive tutorials that can help teach readers on a more intuitive level by allowing readers to modify the tutorial code as they run it. This interactivity is important in that it gives readers the freedom to answer their own questions about the tools. These tutorials should make the on-boarding process easier both for veterans whose job is to teach new researchers, and for the new researchers themselves.

Works Cited

1. Ikram, F. (2019). NirsAutoML: Building an automated classification platform for fNIRS data.
2. Kluyver, T., Ragan-Kelley, B., Fernando Perez, Granger, B., Bussonnier, M., Frederic, J., … Willing, C. (2016). Jupyter Notebooks – a publishing format for reproducible computational workflows. In F. Loizides & B. Schmidt (Eds.), Positioning and Power in Academic Publishing: Players, Agents and Agendas (pp. 87–90).
3. Buntel, E. (2020). Expanding NIRS Auto ML (NAML) to Better Facilitate Neural Data Analysis.