HSST Documentation Manual

HSST Software Package License:

Additional Links:

Source Code:	http://github.com/davidbjanes/hsst
Data Source:	public@davidbjanes.com
Poster:	http://davidbjanes.com/projects
Thesis Manuscript:	http://davidbjanes.com/projects
Presentation Slides:	http://davidbjanes.com/projects
Questions:	public@davidbjanes.com

Getting Started:

Requirements

This software was designed to operate on a Microsoft Windows 7 (x64 bit) operating system, with 8+ GB of RAM, running MATLAB 2013a, 64-bit edition. No other hardware versions or software versions are supported by the author.

Installation

After MATLAB 2013a has been installed, the HSST algorithm may be downloaded as a zip folder or as a cloned folder (using the GIT software), from the GIT hub website at the following url: https://github.com/davidbjanes/hsst. This folder (and all subfolders) must be added to the MATLAB path.

Quick Start

1. Open *hsst.runExampleCode()*
2. Load your neural data into labeled variables (rawWaveform, wf, ts, fw, noiseEstimate)
3. Run hsst.getSortMethods() for complete list of sorting algorithms
4. Set variable *sortMethodObj* to the sort method of your choice
5. Set *sortParameters* to an array of parameter values (leave blank for default values)
6. Run *hsst.sortSnips(…)* to sort waveform snippets OR
   Run *hsst.sortRaw(…)* to sort raw waveform from electrode or a single channel from an array
7. *sortIDs* will give an integer number corresponding the neural ID (one for each waveform)
8. *parameter* will return the parameter value which scored the highest
9. *property* is a structure array of values calculated by HSST (see code for details)
10. Run *hsst.gui(property)* to quickly view and compare the results of HSST!

+HSST Package Description

Internal Packages

+extractorMethod	- contains all spike detection algorithms
+scoreMethod	- contains all parameter scoring algorithms
+sortMethod	- contains all sorting algorithms

Super (Abstract) Classes

@interfacer	- (INCOMPLETE) abstract class for interfacing with a database
@scorer	- abstract class for specifying interface with a scoring class
@sorter	- abstract class for specifying interface with a sorting class
@extractor	- abstract class for specifying interface with a spike detection class

Classes

@optimizer	- class which sorts and scores each data set iteratively
@dataObject	- class which holds all data and meta data
@gui	- classes which generates GUI (takes a dataObject as an input)

Functions

getExtractorMethods()	- returns all valid extactor method objects
getScoreMethods()	- returns all valid score methods objects
getSortMethods()	- returns all valid sort methods objects
isObjectExtractorMethod()	- checks if a class object is a valid extractor method
isObjectScoreMethod()	- checks if a class object is a valid score method
isObjectSortMethod()	- checks if a class object is a valid sorting method
runExampleCode()	- runs example of HSST for demonstration
sortDataObject()	- sorts data placed in a dataObject
sortRaw()	- extracts spike info, and sorts data
sortSnips()	- sorts waveform snippets

In-depth Implementation of HSST

HSST is a spike sorting framework. This implementation utilizes a hierarchical class structure to maintain organization and determine the interfaces between the interchangeable blocks in the software. Let's start with an example.

hsst.runExampleCode

hsst.runExampleCode() is the best place to start to illustrate the operation of HSST. We generate some fake random data as our inputs. *hsst.sortSnips()* and *hsst.sortRaw*() are the main entry points for running the program. *hsst.sortSnips()* as the name implies, sorts already extracted waveform snippets (along with their time points, sampling frequency, etc) while *hsst.sortRaw()* extracts the spike waveforms before sorting. Also passed along is the spike sorting method along with the range of parameter values (specified as an array of values). If *sortParameters* is left blank, the default range of the *sortMethod* is used. All sortMethod's are subclasses of the super abstract class *sorter*.

Once the waveforms are extracted, we build a *dataObject* which contains all the needed variables for sorting. This class holds all the information about the neural data. It is passed between processing classes and updated as needed. It is returned to the user at the end of the processing.

A *scoreMethod* is also chosen (at this point only one method exists, so this value is hard coded. It would be easy to modify this later on if other scoring methods were developed.) All *scoreMethods* are sub classes of the super class *scorer*.

Once the *dataObject* is built, it is passed to *hsst.optimizer()*. The *optimizer* sorts the data iteratively, using each of the parameters given to it. *optimizer* sorts the data in *generateParameterScores()* using the *sortMethod* passed to it. It scores each output of the sorter using the *scoreMethod* and returns the sort output of the highest scoring parameter.

Adding Additional Sort Methods (Algorithms):

Navigate to the ~/+hsst/+sortMethod directory. A folder labeled *@DummySorterExample* shows a blank example of a sorting algorithm. Simply copy the folder (renaming it appropriately, don't forget the folder name, the file name, and the class name). The identically named file inside the folder should have the same name as the folder *../@NewSortingAlgorithm/NewSortingAlgorithm.m*. Inside the *sort* function, you can add whatever algorithm you would like. Don't forget to add this new folder to the

matlab path (otherwise tab complete won't work). *Property* can be anything you'd like; *sort_ids* must be an integer array with length of the number of waveforms (*wf)* and time stamps (*ts)*.

Adding Additional Spike Detection Methods (Algorithms):

Navigate to the ~/+hsst/+extractorMethod directory. A folder labeled *@DummyExtractorExample* shows a blank example of a detection algorithm. Simply copy the folder (renaming it appropriately, don't forget the folder name, the file name, and the class name). The identically named file inside the folder should have the same name as the folder *../@NewExtractionAlgorithm/ NewExtractionAlgorithm.m*. Inside the detection function, you can add whatever algorithm you would like. Don't forget to add this new folder to the matlab path (otherwise tab complete won't work).

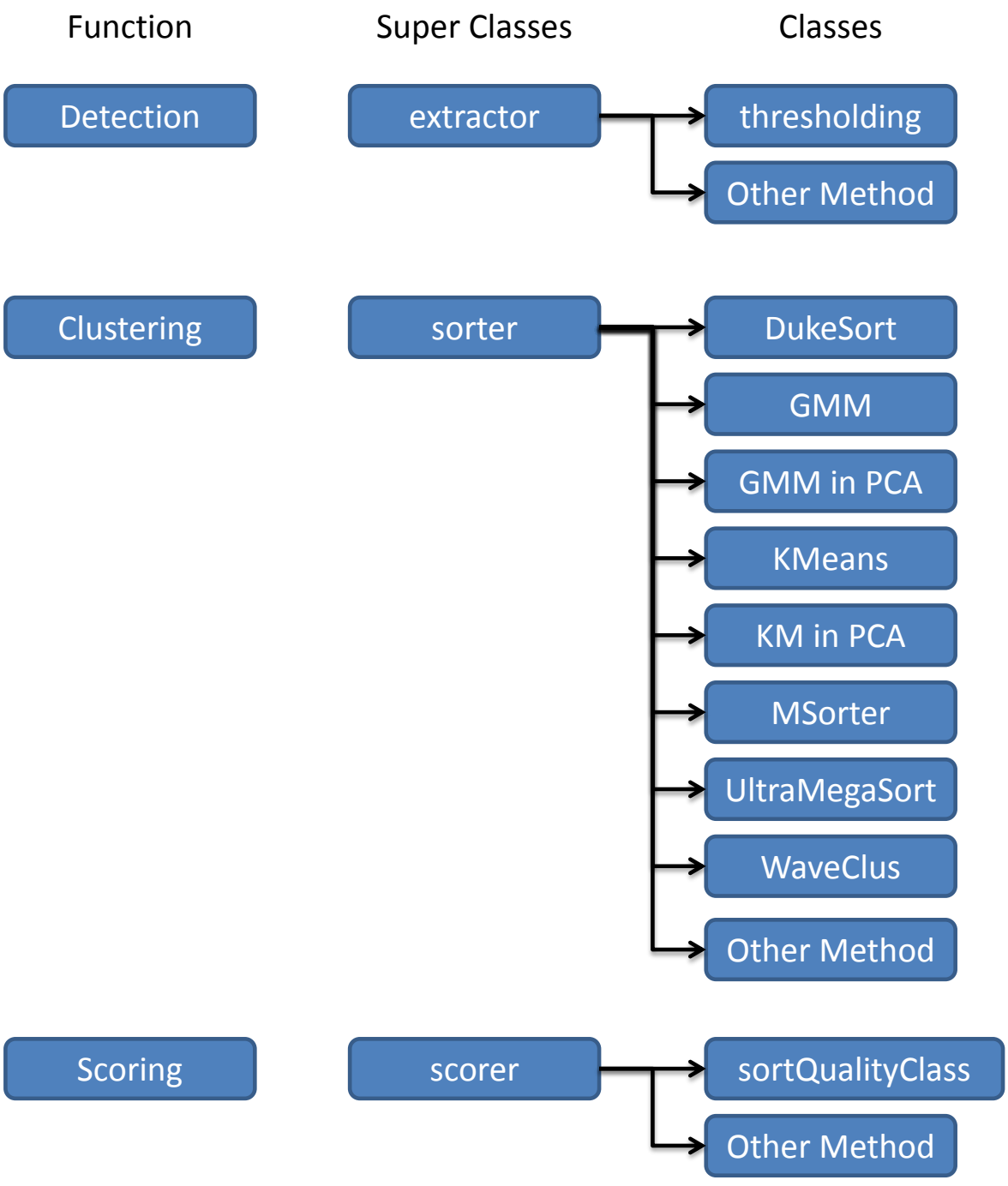
hsst.gui(dataObject)

A GUI is provided to quickly review the results of the algorithm. The *property* value output of the *hsst.sortSnips* is a dataObject. You can run *hsst.gui(property*) to view the results of the HSST parameter selection

Additional Questions

Please send me an email: public@davidbjanes.com with further comments, questions. Feel free to submit revisions, extension or other edits via github. Thanks!

Class Structure



Sample Data Flow