Overview

Unknown Author

November 22, 2013

1 Important Preliminaries:

1). The .kwik format has to be modified so that it can store more than one clustering.

2). Which hashing algorithm? Something from http://docs.python.org/2/library/hashlib.html. We obviously don't need cryptography.

2 Original data files

There is a folder consisting of the three recordings (.dat files) of Mariano Belluscio. One .dat file has been clustered (`n6mab031109.dat`), with method `MKKdistfloat` and has a number of nice clusters out of which we can choose a few suitable donor cells. There are two other recordings `n6mab041109.dat` and `n6mab061109.dat`; these will act as "acceptor" datasets.

3 Hybrid dataset creation

Minimal information necessary for defining a hybrid dataset is a 4-tuple of the form: `(acceptor, donor_id, amplitude, time_series)`, where `acceptor` is the dataset which receives spikes from another analogously recorded dataset `donor` at the times specified by `time_series`. Each element of the 4-tuple may be generated by a range of other parameters, e.g.

1) A rate, `r` could define a `time_series` consisting of regular spiking at `r` Hz. This could be created by a function:

   def: create_regular_resfile(rate, sampling_rate, starttime, endtime, resname): '"Creates a regular .res file with firing rate r Hz samples"'

2) The donor_id could be derived from information, e.g. the 3-tuple (donor,donorcluid,donorcluster) pertaining to the donor dataset (donor), the detection method and clustering algorithm (donorcluid) and cluster (donorcluster) number of the cell that is added to the acceptor dataset.

   def: create_donor_id(donor,donorcluid,donorcluster): "'Outputs donor identity files:

   donor_id = donor_donorcluid_donorcluster, (which typically looks like: n6mab031109_MKKdistfloat_54').

   meanspike_file_id = a file called donor_id.msua.1.

   meanmask_file = a file called donor_id.amsk.1.'"

We can have a folder containing donor_ids, or a list of 3-tuples generating to donor_ids.

We can create a hybrid dataset `D(acceptor, donor_id, amplitude, time_series)` (which we shall abbreviate to `D` when there is no ambiguity) using the Python function:

```
def: create_hybrid_datfile(acceptor, donor_id, amplitude, time_series):

'''This function outputs a raw datafile called,
      Hash(D).dat, in the folder Hash(D)'''

def: convert_to_kwik():
        '''Converts Hash(D).dat to Hash(D).kwd and Hash(D).kwik'''
```

, where Hash(var) is the hash produced from the variables `var` via some hash function.

Question: The raw waveform from one recording on non-relevant channels could contain wildly inappropriate waveforms. I got around this before by only added parts of the wave form corresponding to the .amsk.1 file.

NOTE: `D` implicitly contains the groundtruth times (equivalent to a .res and a .clu file) for the hybrid spike times and hybrid clusters.

4 Running SpikeDetekt with various parameters

We aim to find the optimal detection strategy, but running spikedetekt on Hash(D).dat several times using a variety of parameters, `params`, which we shall denote `p`, resulting in two files in the folder

`Hash(D)_Hash(p):`

`Hash(D)_Hash(p).kwx`

`Hash(D)_Hash(p).kwik`

The feature and mask vectors, corresponding to the .fet and .fmask files are stored in the .kwx file, these will later be required by Masked KlustaKwik.

The parameters are stored in a global Python dictionary of variables which can be accessed by all modules of SpikeDetekt. The default value of these parameters are stored in `spikedetekt/spikedetekt/defaultparameters.py`.

In the first phase we would like to finish testing the effect of varying certain parameters on the quality of spike detection and alignment, e.g. testing two-threshold detection by keeping the upper threshold constant and varying the lower threshold. We can vary one parameter and keep the others constant by specifying custom default parameters in the file `usualparameters.py` . We will specify a 2D detection window, `Sigma` consisting of a minimal time jitter window and a threshold of minimal mask similarity measure (>0.8); a spike will be denoted as "successfully detected" if it lies within this detection window. This will be implemented in some Python functions in `evaluate_detection.py`. This will have two outputs:

1) `detection_statistics(D, p,Sigma)` will be a set of scripts which will measure the efficacy of the detection `p`. e.g. we could have a function: def: test_detection_algorithm(D,p,Sigma): '"Test spike detection algorithm on a hybrid dataset"'

2) A `derived groundtruth(D, p,Sigma)` relative to the detection (i.e. equivalent to a .clu file which is commensurate with the output .res file of detected spikes(those found in the window specified by `Sigma`, which specifies which spikes are background (in cluster 0) and which are hybrid (in clusters 1, 2, ... num_hybrids. This clustering can be stored in Hash(D)*Hash(p)*Sigma.kwik.

5 Supervised Learning

To perform supervised learning in the form of a support vector machine (SVM) on the groundtruth obtained after running SpikeDetekt and applying detection criterion `Sigma`, we use the Python machine learning package, `sklearn`. We will run SVM with a different kernels and their associated parameters, and a grid of class weights. We shall

denote these parameters, s. The set of functions `supervised_learning` should output a set of clusterings from which we obtain an ROC curve, providing an upper bound for performance of any unsupervised algorithm. This set of clusterings will be stored in the file `(D, p,Sigma,s)`.kwik.

6 Clustering using Masked KlustaKwik

Given detection, extraction of features and masking, specified in the files `Hash(D)_Hash(p).kwx`, we can now run KlustaKwik on the .fet and .fmask files obtained from `Hash(D)_Hash(p).kwx`. with a set of parameters which we shall denote `k` which will be defined a dictionary which will output a bash script for running KlustaKwik,

```
def: get_KK_input(D,p,k):
'''Will output the .fet and .fmask files in a folder, CartesianHash_(D,p,k)'''

def: make_KK_runscripts(D,p,k):
'''Produces bash scripts pertaining to parameters k that run KlustaKwik on (D,p) '''

def: make_bash_hash(D,p,k):
'''Outputs the hash of (D,p,k) and the bash script'''
```

This will result in a clustering which will be stored in the file `Hash(D)_Hash(p)_Hash(k).kwx`. An analysis script can be run on `Hash(D)_Hash(p)_Hash(k).kwx` and the associated groundtruth, `Hash(D)_Hash(p)_Sigma.kwik` to obtain the confusion matrix. This will be stored as a pickle or a textfile `Hash(D)_Hash(p)_Hash(k).p`. M# #Supercomputer Scripts will be required for sending jobs to Legion. These jobs will usually be SVM and Masked KlustaKwik.

In []:

In []: