Geometric classification of brain network dynamics via conic derivative discriminants

Matthew F. Singh*, Todd S. Braver, ShiNung Ching
Washington University in St. Louis
*f.singh@wustl.edu

Abstract

This document provides instructions for performing leave-one-out classification using the Conic Derivative Discriminant as described in the paper: Geometric Classification of Brain Network Dynamics via Conic Derivative Discriminants (2018). This function is performed using the included MATLAB function: ConeClass.m and ConeClass_wDeriv.m

1 Requirements

Executing these functions requires MATLAB2016b or later due to the use of implicit expansion for elementwise operations.

2 Input Format

The input to either function consists of a hierarchical cell array. The outermost cell is a $1 \times n_{Class}$ cell containing the structured data for each class. The class data is also a cell-array containing matrices of data for each trial. These matrices are organized as $channel \times time$ (see Figure). As an example, a classification involving two classes (A and B) and three trials each would be run as

$$ConeClass\Big(\Big\{\{A_1, A_2, A_3\}, \{B_1, B_2, B_3\}\Big\}\Big). \tag{1}$$

3 ConeClass vs. ConeClass_wDeriv

The difference between the two functions is whether or not the input has already by converted to L_2-normalized derivatives. There are multiple ways

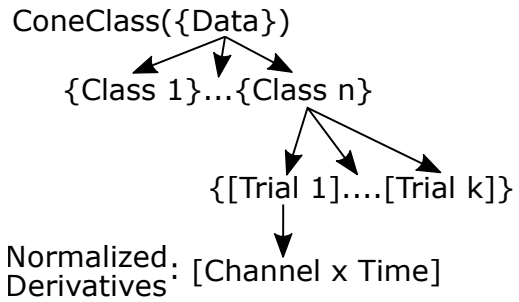

Fig 1. The input data structure consists of nested cells ($\{Class\}\{Trial\}[nChan \times t]$)

of numerically calculating derivatives and users wishing to use a particular method can use ConeClass.m with the data matrices consisting of their pre-calculated derivatives. Otherwise, the function ConeClass_wDeriv.m can be supplied with the regular timeseries and it will numerically calculated the derivatives with finite difference: $dX \approx X(t+1) - X(t)$ and then divide each time point by the L_2 norm across channels. The resulting normalized derivatives are then passed back to ConeClass.m and calculations precede as usual.

4 The Classification Rule: Time-Points

The classification rule is the same as the paper. For an unlabeled time-series $x^{(i(t))}(t)$ with $i(t)$ the to-be-assigned label at each time point we define the classifier as:

$$i(t) = \arg\min_k \left(\frac{\dot{x}(t)^T \Sigma^{-1}_{(\dot{x}_k)} \dot{x}(t)}{det|\Sigma^{-1}_{(\dot{x}_k)}|^{1/n}} \right) \tag{2}$$

Here k is the set of classes and the Σ terms give the covariance of the derivative time series used for training. For leave-one-out cross-validation the training covariance matrix is equal to the mean covariance across training trials (irrespective of the number of entries each). The covariance for each trial is saved in the output field Cov_All which contains one array for each class. Each of these arrays contain the trial covariance for a given class stacked by trial. Thus Cov_Allc(i,j,k) contains the covariance between channels "i" and "j" on the k^{th} trial of class "c". To speed calculations, the code decomposes each function of Σ into its Cholesky form:

$$\frac{\Sigma^{-1}_{(\dot{x}_k)}}{det|\Sigma^{-1}_{(\dot{x}_k)}|^{1/n}} = M_k^T M_k \tag{3}$$

This transformation is done using the eigenvalue decomposition and greatly speeds up calculation. The classification rule for a single time point

simplifies to:

$$i(t) = \arg\min_k \|M_k \dot{x}(t)\|_2^2 \tag{4}$$

These classifications for time-points and can be saved by the optional parameter 'y' as in the call: ConeClass(Data,'y'). This will include the additional field "Rank" in the output. This field is a $1 \times n_{Class}$ cell. Each class cell contains its trials analogous to the input data structure. The matrices, however, correspond to the prediction weights for each time point:

$$Rank\{i\}\{j\}(c,t) = \|M_c \dot{x}_j^i(t)\|_2^2 \tag{5}$$

Here the notation $\dot{x}_j^i(t)$ denotes the derivative of the observation for class i, trial j at timepoint t.

5 Count vs. Confidence for Trial Classification

After classifying time-points the function combines these predictions to classify trials. In the original paper, we collapse across trials by adding the prediction weights:

$$TrialClass = \arg\min_k \sum_{t \in Trial} \|M_k \dot{x}(t)\|_2^2 \tag{6}$$

The overall accuracy across trials for the n-class comparison (between all class types) using this rule is stored in the field Weight_All in which each element is the accuracy for that class. The field Weight_Pair contains the accuracy for each pairwise comparison. Weight_Pair(i,j) contains the accuracy of class "i" in the pairwise classification of "i" vs "j". This matrix is nonsymmetric and the off-diagonals are zero.

Alternatively, trials can be nonparametrically classified according to the maximal number of time-points assigned to each class. The output structure for this rule are the same as for the weight-based classification but with the corresponding fields Count_All and Count_Pair. This variant may be useful for suppressing extreme values.

6 Optional Input Arguments

The ConeClass and ConeClass_wDeriv functions each allow up to three additional arguments. The first argument is whether to save the predictions for each timepoint (see above). By default this option is OFF. The second argument is whether to normalize covariance matrices according to $det|\Sigma\|^{1/n}$. This option is OFF by default. The last optional argument is a threshold

for censoring small eigenvalues when performing matrix inversions and determinants. This value is a relative threshold and has a default value of 10,000 so eigenvalues less than 1/10,000 of the maximal eigenvalue are censored during inversion by default. A value of "p" will censor eigenvalues less than λ_{max}/p with λ_{max} being the maximal eigenvalue in that calculation.