High-dimensional classification for brain decoding

by

Nicole Samantha Croteau
B.Sc., University of Victoria, 2013

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Mathematics and Statistics

High-dimensional classification for brain decoding

by

Nicole Samantha Croteau
B.Sc., University of Victoria, 2013

Supervisory Committee

Dr. Farouk Nathoo, Supervisor
(Department of Mathematics and Statistics)

Dr. Ryan Budney, Departmental Member
(Department of Mathematics and Statistics)

Dr. Julie Zhou, Departmental Member
(Department of Mathematics and Statistics)

Supervisory Committee

Dr. Farouk Nathoo, Supervisor
(Department of Mathematics and Statistics)

Dr. Ryan Budney, Departmental Member
(Department of Mathematics and Statistics)

Dr. Julie Zhou, Departmental Member
(Department of Mathematics and Statistics)

ABSTRACT

Brain decoding involves the determination of a subject's cognitive state or an associated stimulus from functional neuroimaging data measuring brain activity. In this setting the cognitive state is typically characterized by an element of a finite set, and the neuroimaging data comprise voluminous amounts of spatiotemporal data measuring some aspect of the neural signal. The associated statistical problem is one of classification from high-dimensional data. We explore the use of functional principal component analysis, mutual information networks, and persistent homology for examining the data through exploratory analysis and for constructing features characterizing the neural signal for brain decoding. We review each approach from this perspective, and we incorporate the features into a classifier based on symmetric multinomial logistic regression with elastic net regularization. The approaches are illustrated in an application where the task is to infer from brain activity measured with magnetoencephalography (MEG) the type of video stimulus shown to a subject.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

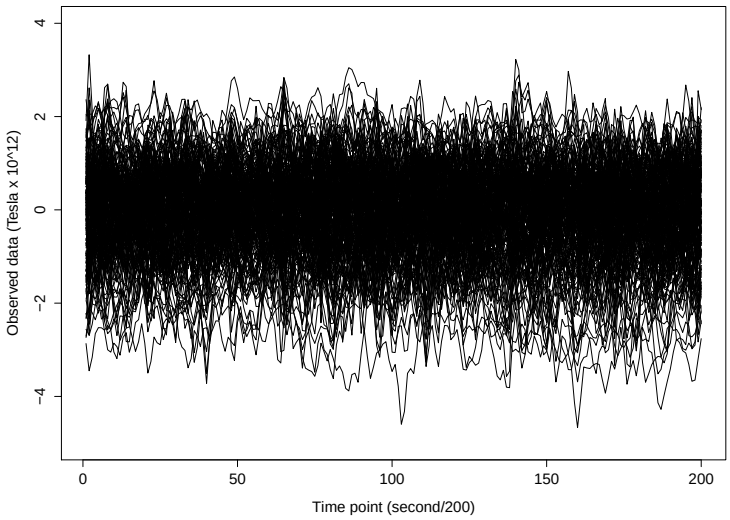
Recent advances in techniques for measuring brain activity through neuroimaging modalities such as functional magnetic resonance imaging (fMRI), electroencephalography (EEG), and magnetoencephalography (MEG) have demonstrated the possibility of decoding a person's conscious experience based only on non-invasive measurements of their brain signals (Haynes and Reese, 2006). Doing so involves uncovering the relationship between the recorded signals and the conscious experience that may then provide insight into the underlying mental process. Such decoding tasks arise in a number of areas, for example, the area of brain-computer interfaces, where humans can be trained to use their brain activity to control artificial devices. At the heart of this task is a classification problem where the neuroimaging data comprise voluminous amounts of spatiotemporal observations measuring some aspect of the neural signal across an array of sensors outside the head (EEG, MEG) or voxels within the brain (fMRI). With neuroimaging data the classification problem can be challenging as the recorded brain signals have a low signal-to-noise ratio; for MEG measurements the magnetic signal produced by neuron activity within the brain is extremely weak, on the order of several tens of femtoTeslas (10^{-15}) to several picoTeslas (10^{-12}), whereas the noise produced by involuntary processes such as heart beats and eye blinks and other factors such as instrument noise and noise from a person thinking about various things at once makes the noise in the recorded brain signals about a factor of 10^3 to 10^6 larger than the brain signals of interest (Hämäläinen et al., 1993). Another challenge of the classification problem with neuroimaging data is the size of the data leads to a high-dimensional problem where it is easy to overfit models to data when

training a classifier. Overfitting will impact negatively on the degree of generalization to new data and thus must be avoided in order for solutions to be useful for practical application.

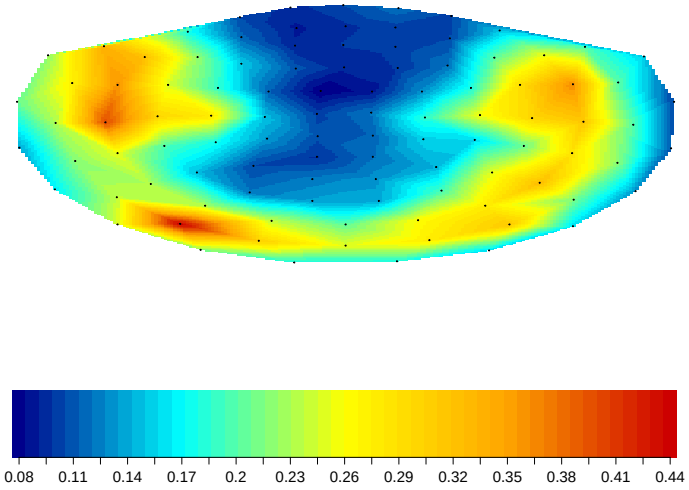
Neuroimaging classification problems have been studied extensively in recent years primarily in efforts to develop biomarkers for neurodegenerative diseases and other brain disorders. A variety of techniques have been applied in this context, including support vector machines (Chapelle et al., 1999), Gaussian process classification (Rasmussen, 2004), regularized logistic regression (Tomioka et al. 2009), and neural networks (Ripely, 1994; Neal and Zhang, 2006). Decoding of brain images using Bayesian approaches is discussed by Friston et al. (2008). While a variety of individual classifiers or an ensemble of classifiers may be applied in any given application, the development of general approaches for constructing features that successfully characterize the signal in functional neuroimaging data is a key open problem. In this thesis we explore the use of some recent approaches developed in statistics and computational topology as potential solutions to this problem. More specifically, we consider how the combination of functional principal component analysis (Ramsay and Silverman, 2005), persistent homology (Carlson, 2009), and network measures of brain connectivity (Rubinov and Sporns, 2010) can be used to (i) explore large datasets of recorded brain activity and (ii) construct features for the brain decoding problem.

The objectives of this thesis are threefold. First, we wish to introduce the brain decoding problem to researchers working in the area of high-dimensional data analysis. This challenging problem serves as a rich arena for applying recent advances in methodology. Moreover, the specific challenges associated with the brain decoding problem (e.g. low signal-to-noise ratio; spatiotemporal data) can help to further motivate the development of new methods. Our second objective is to describe how functional principal component analysis, persistent homology, and network measures of brain connectivity can be used to explore such data and construct features. To our knowledge, functional principal component analysis and persistent homology have not been previously considered as approaches for constructing features for brain decoding.

Our third and final objective is to illustrate these methods in a real application involving MEG data. MEG is a non-invasive neuroimaging technique used to directly measure the magnetic fields generated by neuron activity within the brain. The magnetic fields are captured by multiple sensors at various locations outside the head and are recorded as a collection of time series. In our application the goal is to explore



(a) MEG (magnetic field) signals $Y_{li}(t)$ representing the evoked response collected at $n = 204$ sensor channels.



(b) The variance of the signal (after removal of linear trend) at 102 sensor locations.

Figure 1.1: A single sample from the training data. The map in panel (b) is a two-dimensional projection of the sensor array with the black dots representing the sensor locations. There are two sensors at each location, each oriented differently, and the variance computed from each of the sensors is averaged to obtain a single value (for the purpose of visual summary only).

variability in the brain data and to use the data to infer the type of video stimulus shown to a subject from a 1-second recording obtained from 204 MEG sensor channels with the signal at each channel sampled at a frequency of 200Hz. Each sample thus yields $204 \times 200 = 40800$ observations of magnetic field measurements outside the head. The goal is to decode which of five possible video stimuli was shown to the subject during the recording from these measurements. The data arising from a single sample are shown in Figure 1.1, where panel (a) depicts the brain signals recorded across all sensors during the 1-second recording, and panel (b) depicts the variance of the signal at each location. From Figure 1.1b we see that in this particular sample the stimulus evoked activity in the regions associated with the temporal and occipital lobes of the brain. The entire dataset for the application includes a total of 1380 such samples (727 training; 653 test) obtained from the same subject which together yield a dataset of roughly 6 gigabytes in compressed format. For classification problems we quantify the performance of a classifier based on its misclassification or error rate, that is, the percentage of incorrectly classified samples in the data, with a low error rate indicating a classifier that performs well. To build a classifier and determine its error rate we utilize two sets of data: the *training* samples are seen data whose class labels are known and these samples are used to build the classifier and tune its parameters; the *test* samples are unseen data for which we wish to predict the class labels. It is from these predicted test sample class labels that we obtain the misclassifcation rate for the classifier.

Principal component analysis (PCA) is an exploratory data analysis technique used to emphasize variation and find patterns in high-dimensional data. In essence, PCA transforms a set of possibly correlated finite-dimensional variables into a set of uncorrelated variables, referred to as principal components, that are a linear combination of the original variables by maximizing the variance of the principal components. Typically, the first several components can be used to explain most of the variation in the data and thus PCA is a useful method for reducing the dimension of the data without much loss of information, making PCA an effective tool for data visualization and exploration. Functional principal component analysis (FPCA) is the extension of standard finite-dimensional PCA to the setting where the response variables are functions, a setting referred to as functional data. For clarity, we note here that the use of the word 'functional' in this context refers to functional data as just described, and is not to be confused with functional neuroimaging data which refers to imaging

data measuring the function of the brain. Given a sample of functional observations (e.g. brain signals) with each signal assumed a realization of a square-integrable stochastic process over a finite interval, FPCA involves the estimation of a set of eigenvalue-eigenfunction pairs that describe the major vibrational components in the data. These components can be used to define features for classification through the projection of each signal onto a set of estimated eigenfunctions characterizing most of the variability in the data. This approach has been used recently for the classification of genetic data by Leng and Müller (2005) who use FPCA in combination with logistic regression to develop a classifier for temporal gene expression data.

An alternative approach for exploring the patterns in brain signals is based on viewing each signal obtained at a voxel or sensor as a point in high-dimensional Euclidean space. The collection of signals across the brain then forms a point cloud in this space, and the shape of this point cloud can be described using tools from topological data analysis (Carlson, 2009). In this setting the data are assumed clustered around a familiar object like a manifold, algebraic variety, or cell complex and the objective is to describe (estimate some aspect of) the topology of this object from the data. The subject of persistent homology can be seen as a concrete manifestation of this idea, and provides a novel method for discovering non-linear features in data.

With the same advances in modern computing technology that allow for the storage of large datasets, persistent homology and its variants can be implemented. Persistent homology has been used for medical image processing and features derived from persistent homology have recently been found useful for classification of hepatic (liver) lesions from computed tomography (CT) scans; persistent homology has also been applied for the analysis of structural brain images (Chung et al., 2009; Pachauri et al., 2011). Outside the arena of medical applications, Sethares and Budney (2013) use persistent homology to study topological structures in musical data. Recent work in Heo et al. (2012) connects computational topology with the traditional analysis of variance and combines these approaches for the analysis of multivariate orthodontic landmark data derived from the maxillary complex. The use of persistent homology for exploring structure of spatiotemporal functional neuroimaging data does not appear to have been considered previously.

Another alternative for exploring patterns in the data is based on estimating and summarizing the topology of an underlying network. Networks are commonly used to explore patterns in both functional and structural neuroimaging data. With

the former, the nodes of the network correspond to the locations of sensors/voxels and the links between nodes reflect some measure of dependence between the time series collected at pairs of locations. To characterize dependence between time series, the mutual information, a measure of shared information between two time series, is a useful quantity as it measures both linear and nonlinear dependence (Zhou et al., 2009), the latter being potentially important when characterizing dependence between brain signals (Stam et al., 2003). Given such a network, the corresponding topology can be summarized with a small number of meaningful measures such as those representing the degree of small-world organization (Rubinov and Sporns, 2010). These measures can then be explored to detect differences in the network structure of brain activity across differing stimuli and can be further used as features for brain decoding.

The remainder of the thesis is structured as follows:

Chapter 2 describes the brain decoding problem in detail and presents functional principal component analysis, persistent homology, and mutual information networks as methods for defining features for classification.

Chapter 3 presents an application of the methods outlined in Chapter 2 to the decoding of visual stimuli from MEG data.

Chapter 4 concludes with a brief discussion.

All the R programs and MATLAB programs developed and used in this thesis are given in the Appendix.

Chapter 2

The Neuroimaging Brain Decoding Problem

When an individual experiences some type of stimulus (e.g. visual, emotional) the neurons in certain functional regions of the brain are activated and this activation can be measured using modern neuroimaging methods. By examining the resulting data, brain decoding seeks to uncover the stimulus associated with a person's conscious experience. For example, brain decoding has been popularly applied to the domain of visual perception; in this setting, a subject is shown some type of visual stimulus and their brain activity is recorded, the goal is then to decipher what object the subject was viewing based on their neuroimaging data. Current neuroimaging techniques have shown it is possible to reconstruct a person's cognitive state from non-invasive measurements of their brain activity (Haynes and Reese, 2006).

Conventional location-based neuroimaging approaches seek to uncover a person's cognitive state by establishing which specific regions of the brain are activated by a stimulus. In this setting brain activity is repeatedly measured at multiple locations across the brain but each location is then analyzed *separately*. An alternative approach is to analyze brain activity measured at several locations *simultaneously* which allows the full spatial pattern of brain activity to be considered. This pattern-based approach to brain decoding allows for the possibility that several individual regions of the brain may not carry information about a particular cognitive state, but when taken together, they may prove useful for decoding.

The main focus of this thesis is to provide a solution to the problem of brain de-

coding from neuroimaging data. This chapter begins by describing the brain decoding problem in a statistical context and outlines the different aspects taken into account in search of a solution, including defining a classifier and how to construct features from the data. Specifically, this chapter introduces three methods for constructing features: functional principal component analysis, persistent homology, and mutual information networks.

2.1 Decoding Cognitive States from Neuroimaging Data

Let us assume we have observed functional neuroimaging data $\boldsymbol{Y} = \{y_i(t), i = 1, \ldots, n;\ t = 1, \ldots, T\}$ where $y_i(t)$ denotes the signal of brain activity measured at the i^{th} sensor or voxel at time t. Thus, the neuroimaging data can be represented in the form of a matrix $\boldsymbol{Y} = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n)$, where the recorded brain signal at each location i is represented as $\mathbf{y}_i = (y_i(1), y_i(2), \ldots, y_i(T))'$. We assume that there is a well-defined, but *unknown*, cognitive state corresponding to these data that can be represented by the label $C \in \{1, \ldots, K\}$, where K is known in advance. For example, for the dataset to be considered in Chapter 3, $K = 5$ class labels. Then the decoding problem is that of recovering C from $\boldsymbol{Y}$.

A solution to this problem involves first summarizing $\boldsymbol{Y}$ through an m-dimensional vector of features $\boldsymbol{Y}_f = (Y_{f_1}, \ldots, Y_{f_m})'$ and then applying a classification rule $R^m \to \{1, \ldots, K\}$ to obtain the predicted state. A solution must (i) specify how to construct the features and (ii) define the classification rule. We assume there exists a set of training samples $\boldsymbol{Y}_l = \{y_{li}(t), i = 1, \ldots, n;\ t = 1, \ldots, T\}$, $l = 1, \ldots, L$, with *known* labels C_l for doing this.

To define the classification rule we model the training labels with a multinomial distribution where the class probabilities are related to features through a symmetric multinomial logistic regression (Friedman et al., 2010) having form

$$Pr(C = j|\boldsymbol{Y}_f) = \frac{\exp(\beta_{0j} + \boldsymbol{\beta}_j'\boldsymbol{Y}_f)}{\sum_{k=1}^{K} \exp(\beta_{0k} + \boldsymbol{\beta}_k'\boldsymbol{Y}_f)}, \quad j = 1, \ldots, K \tag{2.1}$$

with parameters $\boldsymbol{\theta} = (\beta_{01}, \boldsymbol{\beta}_1', \ldots, \beta_{0K}, \boldsymbol{\beta}_K')'$, where $\boldsymbol{\beta}_j = (\beta_{1j}, \beta_{2j}, \ldots, \beta_{mj})'$. As the

dimension of the feature vector will be large relative to the number of training samples we estimate $\boldsymbol{\theta}$ from the training data using regularized maximum likelihood. This involves maximizing a penalized log-likelihood

$$\sum_{l=1}^{L} logPr(C = C_l|\boldsymbol{Y}_{lf}) - \lambda \sum_{j=1}^{K} \left(\alpha \left\|\boldsymbol{\beta}_j\right\|_1 + (1-\alpha) \left\|\boldsymbol{\beta}_j\right\|_2^2\right) \tag{2.2}$$

where the likelihood is defined by the symmetric multinomial logistic regression and we incorporate an elastic net penalty (Zou and Hastie, 2006). Optimization is carried using cyclical coordinate descent as implemented in the *glmnet* package in R (Friedman et al., 2010) by maximizing (2.2) along each coordinate direction $\beta_{pj}, 0 \leq p \leq m, 0 \leq j \leq K$, one at a time (assuming α and λ are fixed). The two tuning parameters, $0 \leq \alpha \leq 1$ and $\lambda \geq 0$, weighting the l_1 and l_2 components of the elastic net penalty are chosen using cross-validation over a grid of possible values. Cross-validation is a method for estimating the prediction error of a statistical model over an independent test sample; it is used to choose the optimal values of tuning parameters for a given model (model selection) and, once a model has been specified, to obtain an estimate of the generalization performance of the chosen model (model assessment). Commonly, cross-validation is accomplished by means of k-fold cross-validation whereby the training data are first partitioned into k approximately equal sized segments (folds) and then validation proceeds in rounds, where a single round of cross-validation consists of using $k-1$ data folds to train the model (classifier) and then predicting the output (class labels) of the single remaining fold to obtain an error estimate. The k error estimates are then averaged to give an estimate of the classifier's ability to generalize to new data. Given $\hat{\boldsymbol{\theta}}$ the classification of a new sample $\boldsymbol{Y}^*$ with unknown label is based on computing the estimated class probabilities from (2.1) and choosing the state $C^* \in \{1, \ldots, K\}$ with the highest estimated value.

To define the feature vector $\boldsymbol{Y}_f$ from $\boldsymbol{Y}$ we consider two aspects of the neural signal that are likely important for discriminating cognitive states. The first aspect involves the shape and power of the signal at each location. These are local features computed at each voxel or sensor irrespective of the signal observed at other locations. The variance of the signal computed over all time points is one such feature that will often be useful for discriminating states, as different states may correspond to different locations of activation, and these locations will have higher variability in the signal. The second aspect is the functional connectivity representing how signals at different

locations interact. Rather than being location specific, such features are global and may help to resolve the cognitive state in the case where states correspond to differing patterns of interdependence among the signals across the brain. From this perspective we next briefly describe functional principal component analysis, persistent homology, and mutual information networks as novel approaches for exploring these aspects of functional neuroimaging data, and further how these approaches can be used to define features for classification.

2.2 Functional Principal Component Analysis

Let us fix a particular location i of the brain or sensor array. At this specific location we observe a sample of curves $y_{li}(t)$, $l = 1, \ldots, L$, where the size of the sample corresponds to that of the training set. We assume that each curve is an independent realization of a square-integrable stochastic process $y_i(t)$ on $[0, T]$ with mean $E[y_i(t)] = \mu_i(t)$ and covariance $Cov[y_i(t), y_i(s)] = G_i(s, t)$. The process can be written in terms of the Karhunen-Loève representation (Leng and Müller, 2005)

$$y_i(t) = \mu_i(t) + \sum_m \epsilon_{mi}\rho_{mi}(t) \tag{2.3}$$

where $\{\rho_{mi}(t)\}$ is a set of orthogonal functions referred to as the functional principal components (FPCs) with corresponding coefficients

$$\epsilon_{mi} = \int_0^T (y_i(t) - \mu_i(t))\rho_{mi}(t)dt \tag{2.4}$$

with $E[\epsilon_{mi}] = 0$, $Var[\epsilon_{mi}] = \lambda_{mi}$ and the variances are ordered so that $\lambda_{1i} \geq \lambda_{2i} \geq \cdots$ with $\sum_m \lambda_{mi} < \infty$. The total variability of process realizations about $\mu_i(t)$ is governed by the random coefficients ϵ_{mi} and in particular by the corresponding variance λ_{mi}, with relatively higher values corresponding to FPCs that contribute more to this total variability.

Given the L sample realizations, the estimates of $\mu_i(t)$ and of the first few FPCs can be used to explore the dominant modes of variability in the observed brain signals at location i. The mean curve is estimated simply as $\hat{\mu}_i(t) = \frac{1}{L}\sum_{l=1}^{L} y_{li}(t)$ and from this the covariance function $G_i(s, t)$ is estimated $\hat{G}_i = \hat{Cov}[y_i(s_k), y_i(s_l)]$ us-

ing the empirical covariance over a grid of points $s_1, \ldots, s_S \in [0, T]$. The FPCs are then estimated through the spectral decomposition of $\hat{G}_i$ (see e.g. Ramsay and Silverman, 2005) with the eigenvectors yielding the estimated FPCs evaluated at the grid points, $\hat{\boldsymbol{\rho}}_{mi} = (\hat{\rho}_{mi}(s_1), \ldots, \hat{\rho}_{mi}(s_S))'$, and the corresponding eigenvalues being the estimated variances $\hat{\lambda}_{mi}$ for the coefficients ϵ_{mi} in (2.3). The fraction of the sample variability explained by the first M estimated FPCs can then be expressed as $FVE(M) = \sum_{m=1}^{M} \hat{\lambda}_{mi} / \sum_m \hat{\lambda}_{mi}$ and this can be used to choose a nonnegative integer M_i so that the predicted curves

$$\hat{y}_{li}(t) = \hat{\mu}_i(t) + \sum_{m=1}^{M_i} \hat{\epsilon}_{lmi} \hat{\rho}_{mi}(t)$$

explain a specified fraction of the total sample variability. We note that in producing the predicted curve a separate realization of the coefficients ϵ_{mi} from (2.3) is estimated from each observed signal using (2.4) and, for a given m, the estimated coefficients $\hat{\boldsymbol{\epsilon}}_{mi} = \{\hat{\epsilon}_{lmi},\, l = 1, \ldots, L\}$ are referred to as the order-m FPC scores which represent between subject variability in the particular mode of variation represented by $\hat{\rho}_{mi}(t)$. The scores are thus potentially useful as features for classification.

We compute the FPC scores $\hat{\boldsymbol{\epsilon}}_{mi}$, $m = 1, \ldots, M_i$ separately at each location $i = 1, \ldots, n$. For a given location the number of FPCs, M_i, is chosen to be the smallest integer such that the $FVE(M_i) \geq 0.9$. Thus the number of FPCs, M_i, will vary across locations but typically only a small number will be required. Locations requiring a relatively greater number of FPCs will likely correspond to locations where the signal is more complex. The total number of features introduced by our application of FPCA for brain decoding is then $\sum_{i=1}^{n} M_i$.

2.3 Persistent Homology

Let us now fix a particular sample l from the training set and consider the collection of signals, $y_{li}(t)$, observed over all locations $i = 1, \ldots, n$ for that sample. Each signal is observed over the same set of T equally-spaced time points $\boldsymbol{Y}_{li} = (y_{li}(1), \ldots, y_{li}(T))'$ and can thus be considered a point in R^T. The sample of signals across the brain/sensors then forms a point cloud in R^T. For example, the single sample depicted in Figure 1.1a represents a cloud of $n = 204$ points in R^{200}. Using tools

from topological data analysis we aim to identify topological structures associated with this point cloud and to use such structures as features for brain decoding.

As a metric inducing the topology of the point cloud we require a measure of statistical dependence that will collate both correlated and anti-correlated signals. We therefore employ the absolute Pearson correlation distance metric $D(\boldsymbol{Y}_{li}, \boldsymbol{Y}_{lj}) = 1 - \rho(\boldsymbol{Y}_{li}, \boldsymbol{Y}_{lj})^2$ where $\rho(\boldsymbol{Y}_{li}, \boldsymbol{Y}_{lj})$ is the sample correlation between signals at locations i and j. We focus specifically on persistent homology, which attempts to identify the connected components, loops, and voids of an associated manifold that we assume the point cloud has been sampled from. The general idea is to approximate the manifold using a simpler object, a simplicial complex, for which the homology (a characterization of the holes in the shape) may be calculated. A sequence of such approximations covering a range of scales is considered and the features that persist over a large range are considered as intrinsic to the data. We provide here only a general description that emphasizes basic concepts and intuition for the construction of features for brain decoding. A more detailed but still gentle introduction to persistent homology, including the required definitions and results from simplicial homology theory and group theory, is provided by Zhu (2013).

Given the point cloud of n signals and the metric based on correlation distance, we consider a covering of the points by balls of radius ϵ, and for any such covering, we associate a simplicial complex for which the homology classes can be calculated. The p-th Betti number, Betti_p, can be thought of as representing the number of p-dimensional holes, which for $p = 0, 1, 2$ corresponds to the *number of connected components*, *loops* and *voids*, respectively. The value of ϵ is then varied over many possible values creating a filtration (an increasing sequence of simplicial complexes). The growing radius ϵ corresponds to allowing signals with smaller values of the squared-correlation to be linked together to form simplices in the simplicial complex. The homology classes are calculated at each value of ϵ to determine how the system of holes changes. Persistent topological features remain over a relatively large range of ϵ values and are thus considered as signal in the data.

The nature of change with respect to each dimension p can be depicted graphically using a barcode plot, a plot that tracks the birth and death of holes as ϵ varies. Features in the barcode that are born and then quickly die are considered topological noise, while features that persist are considered indicative of signal in the underlying topology. If the barcode plot of dimension $p = 0$ reveals signal and the higher-

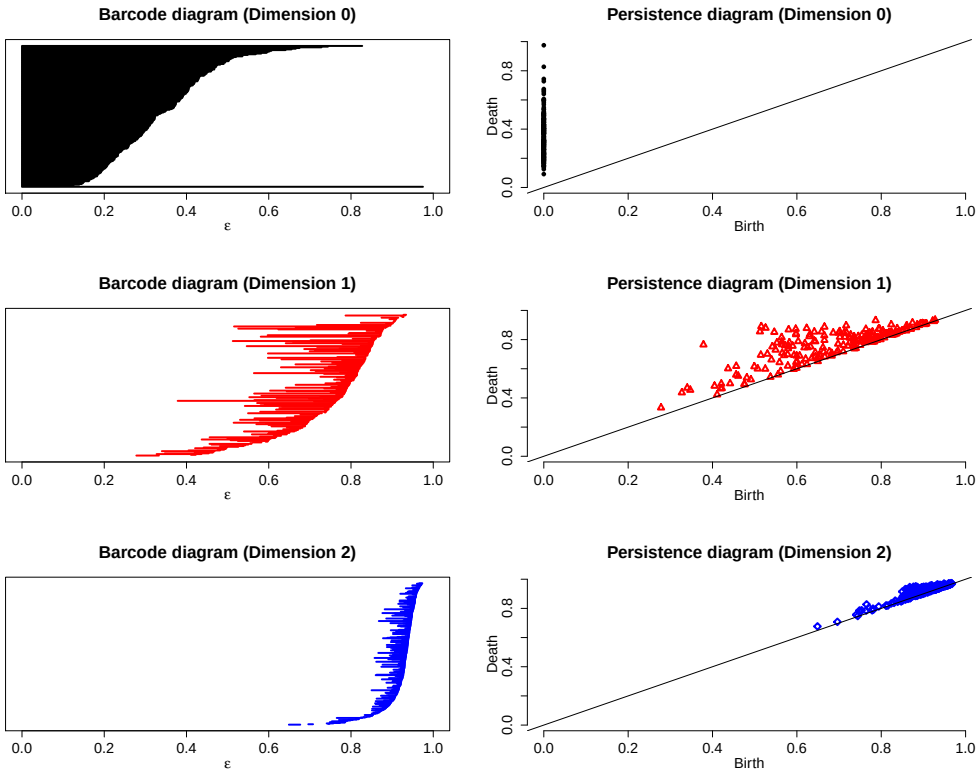


Figure 2.1: Persistent homology computed for the single training sample depicted in Figure 1.1a. The first column displays the barcodes for dimension $p = 0, 1, 2$ in each of the three rows respectively, and the second column displays the corresponding persistence diagrams.

dimensional barcodes do not, the data are clustered around a metric tree. If both the $p = 0$ and $p = 1$ barcodes reveal signal and the $p = 2$ barcode plot does not, the data are clustered around a metric graph. A metric graph is indicative of multiple pathways for signals to get between two sensors/voxels. For barcodes of dimension $p > 1$ the details can be rather subtle to sort through

For the sample considered in Figure 1.1a, the barcodes for each dimension $p = 0, 1, 2$ are depicted in the first column of Figure 2.1. For a given barcode plot, the Betti number for fixed ϵ is computed as the number of bars above it. For $p = 0$ (Figure 2.1, first row and first column), $\text{Betti}_0 = 204$ connected components are born at $\epsilon = 0$ corresponding to each of the MEG sensors. Betti_0 decreases rapidly as ϵ increases and it appears that between two to four connected components persist over a wide range of ϵ values. The barcode plot for dimension $p = 1$ (Figure 2.1, second row and first column) also appears to have features that are somewhat significant,

but the $p = 2$ barcodes are relatively short, indicating noise. Taken together this can be interpreted as there being many loops in the point cloud. The data resemble a metric graph with some noise added. An equivalent way to depict the persistence of features is through a persistence diagram, which is a scatter plot comparing the birth and death ϵ values for each hole. The persistence diagrams corresponding to each barcode are depicted in the second column of Figure 2.1.

As for interpretation in the context of functional neuroimaging data, the number of connected components (Betti_0) represents a measure of the overall connectivity or synchronization between sensors, with smaller values of Betti_0 corresponding to a greater degree of overall synchrony. We suspect that the number of loops (Betti_1) corresponds to the density of 'information pathways' with higher values corresponding to more complex structure having more pathways. The number of voids (Betti_2) may be related to the degree of segregation of the connections. If a void was to persist through many values of ϵ, then we may have a collection of locations/sensors that are not communicating. Thus the larger the value of Betti_2, the more of these non-communicative spaces there may be.

For each dimension $p = 0, 1, 2$ we construct features for classification by extracting information from the corresponding barcode by considering the persistence of each hole appearing at some point in the barcode. This is defined as the difference between the corresponding death and birth ϵ values for a given hole. This yields a sample of persistence values for each barcode. Summary statistics computed from this sample are then used as features for classification. In particular, we compute the total persistence, PM_p, which is defined as one-half of the sum of all persistence values, and we also compute the variance, skewness, and kurtosis of the sample leading to additional features denoted as PV_p, PS_p, PK_p, respectively. In total we obtain twelve global features for brain decoding from persistent homology.

It is worth noting that the persistence, and the summary statistics derived from the persistence values, represent only one way of summarizing the topological information encoded in the barcodes. In future work it may be worth examining persistence landscapes (Bubenik, 2012) as mediums for summarizing the topological information represented by the barcodes as these landscapes may lead to more discriminative features for the classification problem.

2.4 Mutual Information Networks

Let us again fix a particular sample l from the training set and consider the collection of signals, $y_{li}(t)$, observed over all locations $i = 1, \ldots, n$ for the given sample. For the moment we will suppress dependence on training sample l and let $\boldsymbol{Y}_i = (Y_i(1), \ldots, Y_i(T))'$ denote the time series recorded at location i. We next consider a graph theoretic approach that aims to characterize the global connectivity in the brain with a small number of neurobiologically meaningful measures. This is achieved by estimating a weighted network from the time series where the sensors/voxels correspond to the nodes of the network and the links $\hat{\boldsymbol{w}} = (\hat{w}_{ij})$ represent the connectivity, where $\hat{w}_{ij}$ is a measure of statistical dependence estimated from time series $\boldsymbol{Y}_i$ and $\boldsymbol{Y}_j$.

As a measure of dependence to define the network links, we consider the mutual information which quantifies the shared information between two time series and measures both linear and nonlinear dependence. The coherence between $\boldsymbol{Y}_i$ and $\boldsymbol{Y}_j$ at frequency λ is a measure of correlation in frequency and is defined as

$$coh_{ij}(\lambda) = \frac{|f_{ij}(\lambda)|^2}{f_i(\lambda) \times f_j(\lambda)}$$

where $f_{ij}(\lambda) = \sum_{h=-\infty}^{\infty} Cov_{ij}(t, t+h) \cdot exp(-2\pi i\ \lambda h)$ is the cross-spectral density between $\boldsymbol{Y}_i$ and $\boldsymbol{Y}_j$ and $f_j(\lambda) = \sum_{h=-\infty}^{\infty} Cov_j(t, t+h) \cdot exp(-2\pi i\ \lambda h)$, $f_i(\lambda) = \sum_{h=-\infty}^{\infty} Cov_i(t, t+h) \cdot exp(-2\pi i\ \lambda h)$, are the corresponding spectral densities for each process (see e.g. Shumway and Stoffer, 2011). The mutual information within frequency band $[\lambda_1, \lambda_2]$ is then

$$\delta_{ij} = -\frac{1}{2\pi} \int_{\lambda_1}^{\lambda_2} \log(1 - coh_{ij}(\lambda)) d\lambda$$

and the network weights are defined as $w_{ij} = \sqrt{1 - \exp(-2\delta_{ij})}$ which gives a measure of dependence lying in the unit interval (Joe, 1989). The estimates $\hat{\boldsymbol{w}}$ are based on frequency values $\lambda_1 = 0$ $\lambda_2 = 0.5$, and computed using the MATLAB toolbox for functional connectivity (Zhou et al., 2009). After computing the estimates of the network matrices we retained only the top 20% strongest connections and remove the remaining connections from the network.

We summarize the topology of the network obtained from each sample with seven

graph-theoretic measures, each of which can be expressed explicitly as a function of the network weights $\hat{\boldsymbol{w}}$ (see e.g. Rubinov and Sporns, 2010):

1. Characteristic path length: the average shortest path between all pairs of nodes.
2. Global efficiency: the average inverse shortest path length between all pairs of nodes.
3. Local efficiency: global efficiency computed over node neighbourhoods.
4. Clustering coefficient: an average measure of the prevalence of clustered connectivity around individual nodes.
5. Transitivity: a robust variant of the clustering coefficient.
6. Modularity: degree to which the network may be subdivided into clearly delineated and non-overlapping groups.
7. Assortativity coefficient: correlation coefficient between the degrees of all nodes on two opposite ends of a link.

In computing the measures, the distance between any two nodes is taken as $\hat{w}_{ij}^{-1}$. The seven measures are computed for each training sample and used as global features for brain decoding.

In summary, this chapter has provided an introduction to the brain decoding problem for functional neuroimaging data and has outlined a potential solution utilizing an elastic net regularized symmetric logistic classifier. Additionally, it has given an overview of three methods for feature construction based on FPCA, persistent homology, and mutual information networks. The next chapter presents an application of these methods to the decoding of visual stimuli from MEG data.

Chapter 3

Brain Decoding from MEG Data : A Case Study

The purpose of this chapter is to give an example application of the methods described in Chapter 2 to a real-life neuroimaging dataset. We begin with a description of the dataset followed by details as to how features were derived from the data for each method under consideration. This is followed by a description of the cross-validation algorithm used to estimate the tuning parameters of the classifier with elastic net penalty, and finally we present the results of our methodology for this application.

3.1 MEG Mind Reading Dataset

In 2011 the International Conference on Artificial Neural Networks (ICANN) held an MEG mind reading contest sponsored by the PASCAL2 Challenge Programme. The challenge task was to infer from brain activity measured with MEG the type of a video stimulus shown to a subject. The experimental paradigm involved one male subject who watched alternating video clips from five video types while MEG signals were recorded at $n = 204$ sensor channels covering the scalp. The different video types are:

1. Artificial: screen savers showing animated shapes or text.
2. Nature: clips from nature documentaries, showing natural scenery like mountains or oceans.

3. Football: clips taken from (European) football matches of Spanish La Liga.
4. Mr. Bean: clips from the episode *Mind the Baby, Mr. Bean* of the Mr. Bean television series.
5. Chaplin: clips from the *Modern Times* feature film, starring Charlie Chaplin.

The experiment consisted of two separate recording sessions that took place on consecutive days. The experiment, where the subject watched a series of video clips without audio while his brain signals were recorded, was conducted in the same manner during both recording sessions. Specifically, the video clips were grouped in five blocks, where videos from the first four blocks contained a mixture of alternating short video clips from the 'Artificial', 'Nature', and 'Football' videos. Within each block the clips were played for 6 to 26 seconds and were separated by a 5-second rest period during which the subject viewed a crosshair in the visual field. The last block contained two longer clips from the two video stimuli with a storyline. In this block, the subject viewed clips of approximately 10 minutes from both an episode of Mr. Bean and a Charlie Chaplin film. From this experiment, the organizers released a series of 1-second MEG recordings in random order which were downsampled to 200Hz. The data are available from http://www.cis.hut.fi/icann2011/meg/measurements.html[1].

A single 1-second recording is depicted in Figure 1.1a, and the data comprise a total of 1380 such recordings. Of these, 677 recordings are labelled training samples from the first day of the experiment and 653 are unlabelled test samples from the second day of the experiment. Thus aside from the challenge of decoding the stimulus associated with test samples an additional challenge arises in that the training and test sets are from different days, leading to a potential domain shift problem. To aid contestants with this problem the organizers released a small additional set of 50 labelled training samples from day two. The objective was to use the 727 labelled training samples to build a classifier, and the submissions were judged based on the overall accuracy rate for decoding the stimulus of the test samples. The accuracies of the solutions submitted to the competition are given in Table 3.1. The overall winning team obtained an accuracy rate of 68.0%, which was followed by 63.2% for the second place entry, and the remaining scores ranged from 62.8% to 24.2%. It is

[1]We note that there are six types of signals available in the data files: the raw unfiltered MEG signal and five signals resulting from applying a filter bank to the raw signals with frequencies centered around 2, 5, 10, 20, and 35 Hz. For the purposes of this thesis we only consider the unfiltered MEG signals.

Table 3.1: The results of the ICANN 2011 MEG mind reading competition.

Team	Accuracy
Huttunen et al.	68.0%
Santana et al.	63.2%
Jylnki et al.	62.8%
Tu & Sun	62.2%
Lievonen & Hytyniemi	56.5%
Tu & Sun (2)	54.2%
Olivetti & Melchiori	53.9%
Van Gerven & Farquhar	47.2%
Grozea	44.3%
Nicolaou	24.2%

noted that the chance level for a classifier learned on this data is 23.0%, resulting from assigning all test samples to the stimulus class associated with the largest number of training samples. Full details of the competition and results are available in Klami et al. (2011). Following the competition, the labels for the 653 test samples were also released.

Our objective is to apply the techniques described in this thesis to the ICANN MEG dataset and to compare the resulting decoding accuracy rates to those obtained in the actual competition. In conducting our analysis all of the competition rules were followed and the test data were only used to evaluate our approach, as in the competition.

3.2 Feature Construction for Classification

Examination of the training data reveals the detrended variance (variance of signal after removal of linear trend) at each sensor to be an important feature for discriminating the stimuli. This is as expected (see discussion in Chapter 2) and so all classifiers we consider include this feature. Experimentation (using only the training data) with classifiers excluding the detrended variance indicated that this is by far the most important feature and the predicted accuracy rates we obtain from cross-validation drop significantly when this feature is excluded. In Figure 3.1 we illustrate the average spatial variation of this feature for each of the five stimuli. Differing patterns are seen for each class. For example, in the 'Chaplin' class, the signal exhibits

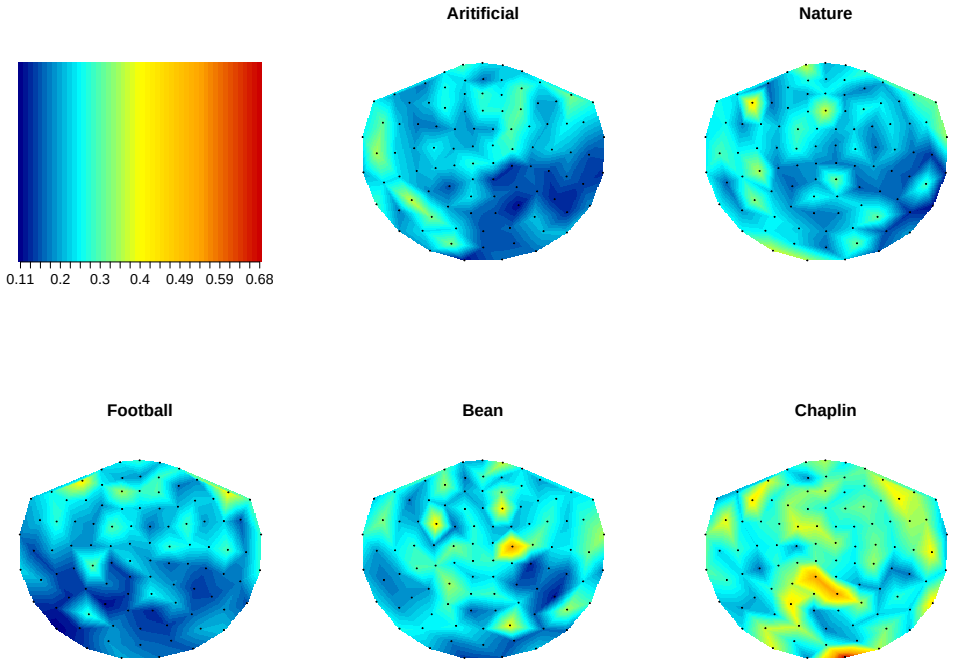
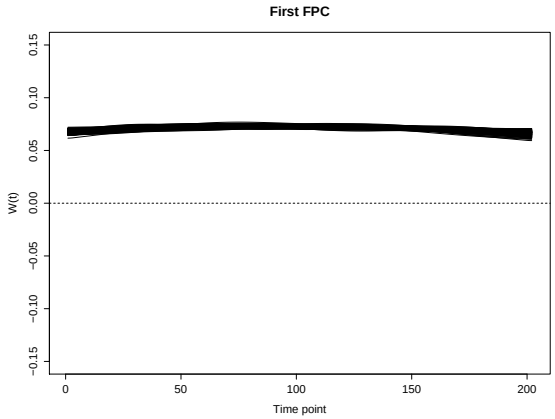


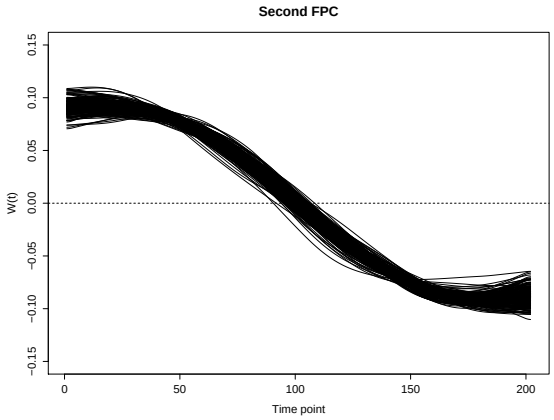
Figure 3.1: Spatial variation of the detrended variance by stimulus class. Each map is a two-dimensional projection of the sensor array with the black dots representing the sensors. At each sensor we fit a linear regression on time point and compute the variance of the residuals as the feature. There are 2 sensors (each oriented differently) at each of 102 locations. For the purpose of visual summary, we average the two variance measures for each location and then further average across all training samples within a given stimulus class. We then map the resulting averaged measures across the scalp.

greatest power in sensors representing the occipital and parietal lobes; whereas for the 'Football' class we see the greatest power in sensors representing the left and right frontal lobes. Including the detrended variance at each sensor yields 204 features to be added to the classifier.

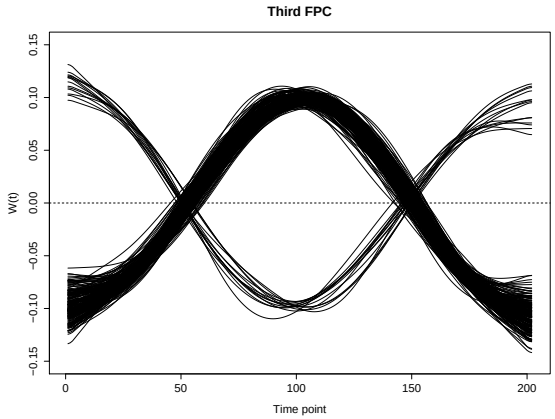
To derive additional features we applied FPCA to all of the training samples separately at each sensor. The functional principal components and the associated FPC scores are computed using the *fda* package in R (Ramsay and Silverman, 2005). Figure 3.2 shows the first three functional principal components. The first FPC, depicted in Figure 3.2a, seems to capture the overall level of the signal. The second FPC, depicted in Figure 3.2b, appears to represent an overall trend and the third FPC, depicted in Figure 3.2c, is a mode of variation having a 'U' or an inverted



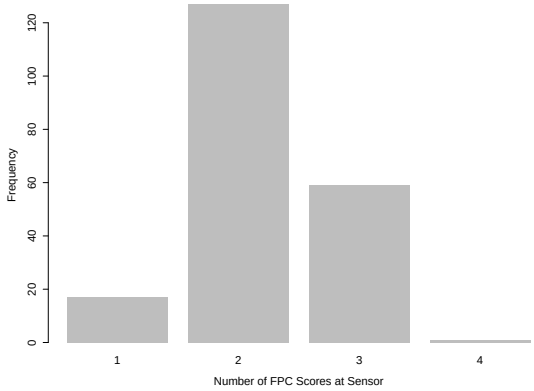
(a) The first FPC at each sensor.



(b) The second FPC at each sensor.



(c) The third FPC at each sensor.



(d) The distribution of the smallest number of FPCs required to explain at least 90% of the variance at each sensor.

Figure 3.2: FPCA applied to the training data.

'U' shape. At each sensor, we included as features the minimum number of FPC scores required to explain 90% of the variability at that sensor across the training samples. The distribution of the number of scores used at each sensor is depicted in Figure 3.2d. At most sensors either $M_i = 2$ or $M_i = 3$ FPC scores are used as features, and overall, FPCA introduces 452 features. The spatial variability of the first FPC score is depicted in Figure 3.3. Differing spatial patterns across all of the stimuli are visible, in particular for the 'Chaplin' class, which tends to have elevated first FPC scores at many sensors.

Persistent homology barcodes of dimension $p = 0, 1, 2$ were computed using the

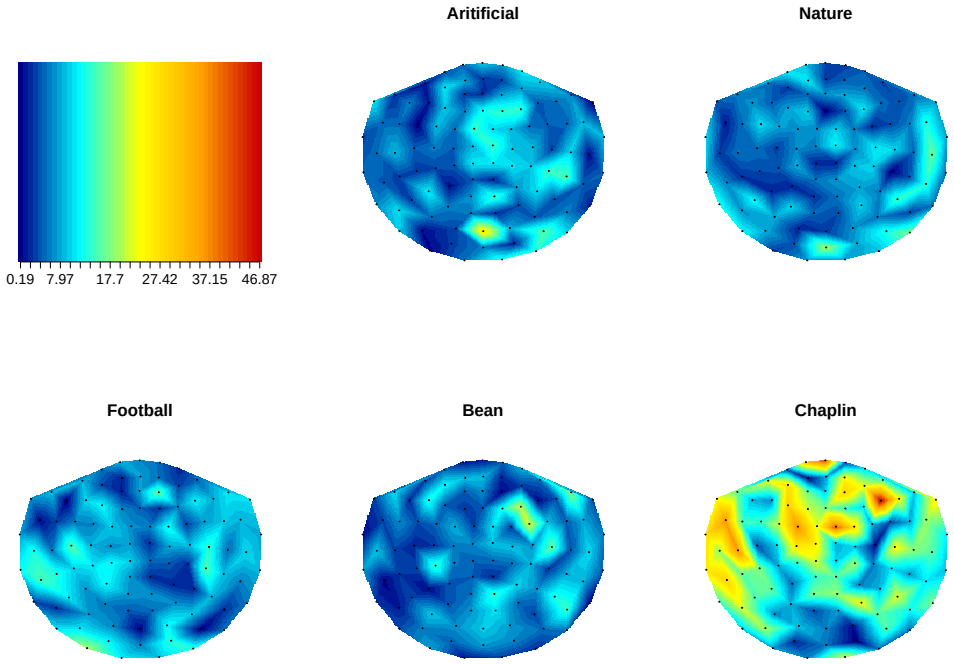


Figure 3.3: Spatial variation of the first FPC score by stimulus class. Each map is a two-dimensional projection of the sensor array with the black dots representing the sensor locations. There are 2 sensors (each oriented differently) at each of 102 locations. For the purpose of visual summary, we average the absolute value of the 2 scores at each location and then further average across all training samples within a given stimulus class. We then map the resulting averaged measures across the scalp.

TDA package in R (Fasy et al., 2014) for all training samples and the 12 summary features PM_p, PV_p, PS_p, PK_p, $p = 0, 1, 2$ were extracted from the barcodes. To determine the potential usefulness for classification we compared the mean of each of these features across the five stimuli classes using one-way analysis of variance. Table 3.2 reports the p-values obtained from this analysis. In most cases the p-value corresponding to the null hypothesis of equality of the mean across all groups was less than 0.05, with the exception of PK_0 (p-value = .364) and PM_1 (p-value = .343). This indicates that, individually, the PK_0 and PM_1 features do not show much promise of being useful for classification purposes, however, this preliminary analysis leads us to believe that the persistent homology features *as a whole* may be useful for differentiating the stimuli classes of these data.

Mutual information weighted networks were also computed for each training sam-

Table 3.2: Results of comparing the distribution of persistent homology features across stimuli groups. We report the p-values produced from a one-way ANOVA testing the hypothesis of equality of persistent homology feature means across stimuli groups. Values less than .05 are highlighted in bold.

Feature	p-value
PM_0	**.031**
PM_1	.343
PM_2	**.039**
PV_0	**.005**
PV_1	**.001**
PV_2	$<$ **.001**
PS_0	$<$ **.001**
PS_1	**.014**
PS_2	**.002**
PK_0	.364
PK_1	**.035**
PK_2	**.011**

Table 3.3: Results of comparing the distribution of network features across stimuli groups. We report the p-values produced from a one-way ANOVA testing the hypothesis of equality of network feature means across stimuli groups.

Network measure	p-value
Characteristic path length	$<$ .001
Global efficicency	$<$ .001
Local efficiency	$<$ .001
Clustering coefficient	$<$ .001
Transitivity	$<$ .001
Modularity	$<$ .001
Assortativity coefficient	$<$ .001

ple using the MATLAB toolbox for functional connectivity (Zhou et al., 2009) and the seven graph theory measures discussed in Section 2.4 were calculated. Analysis of variance comparing the mean of each graph measure across stimuli classes resulted in p-values less than 0.001 for all seven features, as indicated in Table 3.3. This initial analysis indicates that both persistent homology and network features, particularly the latter, may be useful for discriminating the stimuli for these data.

3.3 Performance Assessment

We considered a total of seven classifiers based on the setup described in Chapter 2 each differing with respect to the features included. The features included in each of the classifiers are indicated in Table 3.4. The simplest classifier included only the detrended variance (204 features) and the most complex classifier included the detrended variance, FPCA scores, persistent homology statistics, and graph theory measures (675 features).

Table 3.4: Results from the brain decoding competition dataset. Baseline test accuracy is 23.0% (chance level); competition winners achieved 68.0% and second place was 63.2%. Note that 'PH' refers to the 12 features derived using persistent homology.

Classifier	# of Features	CV Predicted Accuracy	Test Accuracy
1 - Detrended variance	204	60.90%	61.26%
2 - Detrended variance + FPCA	656	60.90%	65.54%
3 - Detrended variance + Network Features	211	60.46%	61.41%
4 - Detrended variance + PH	216	60.44%	61.10%
5 - **Detrended variance + FPCA + Network Features**	663	**61.68%**	**66.46%**
6 - Detrended variance + FPCA + PH	668	60.72%	64.01%
7 - Detrended variance + FPCA + Network Features + PH	675	61.68%	65.24%

As discussed in Chapter 2, the regression parameters $\boldsymbol{\theta}$ are estimated by maximizing the log-likelihood of the symmetric multinomial logistic regression subject to an elastic net penalty (2.2). The elastic net penalty is a mixture of ridge and lasso penalties and has two tuning parameters: $\lambda \geq 0$ a complexity parameter, and $0 \leq \alpha \leq 1$ a parameter balancing the ridge ($\alpha = 0$) and lasso ($\alpha = 1$) components. We choose values for these tuning parameters using cross-validation based on a nested cross-validation scheme similar to that proposed in Huttunen et al. (2011) that emphasizes the 50 labelled day two samples for obtaining error estimates. A description of this cross-validation algorithm is given below.

Table 3.5: Computation time required to obtain a performance estimate for the classifiers based on a 200-fold cross-validation of the tuning parameters using parallel computing.

Classifier	Computation Time
1 - Detrended variance	3.61 hours
2 - Detrended variance + FPCA	6.32 hours
3 - Detrended variance + Network Features	3.61 hours
4 - Detrended variance + PH	3.65 hours
5 - Detrended variance + FPCA + Network Features	6.51 hours
6 - Detrended variance + FPCA + PH	6.60 hours
7 - Detrended variance + FPCA + Network Featues + PH	6.69 hours

3.3.1 Nested Cross-Validation Algorithm

We consider a sequence of possible values for α lying in the set $\{0, 0.1, 0.2, \ldots, 1.0\}$ and fix α at one such value. With the given value of α fixed, we perform a 200-fold cross-validation with the 727 training samples[2]. In each fold, the training data consists of all 677 samples from day one and a random sample of 25 of the 50 labelled day two samples. The remaining labelled day two samples are set aside as a validation set for the given fold. Within this fold, the 677+25 = 702 samples in the current training set are subjected to another 5-fold cross-validation over a sequence of λ values to obtain an optimal λ value for the given α *and* training set. The resulting model is then used to classify the 25 validation samples resulting in a performance estimate $\epsilon_{\alpha,j}$ corresponding to the j^{th} fold, $j = 1, \ldots, 200$. The overall performance estimate for a given α is then obtained as the mean over the 200 folds $\epsilon_\alpha = \frac{1}{200}\sum_{j=1}^{200}\epsilon_{\alpha,j}$. This procedure is repeated for all α in $\{0, 0.1, ..., 1.0\}$. The optimal value for the tuning parameter α is that which corresponds to the smallest error ϵ_α.

This entire procedure is repeated for each feature set $\boldsymbol{Y}_f,\ f = 1, \ldots, F$, to produce an optimal α_f value and corresponding error ϵ_{α_f} for each classifier. These errors, $\epsilon_{\alpha_f},\ f = 1, \ldots, F$, are the cross-validated predicted accuracy rates for the classifiers. We choose the classifier/feature set with the highest predicted accuracy rate to be the optimal classifier and the optimal value for λ is again chosen by 5-fold cross-validation as done previously, but now using all of the 727 training samples from both days.

The cross-validation procedure can be quite computationally expensive; estimating the model parameters for just one test split can take 10 to 40 minutes. Thus a

[2]Note: The feature matrix is assumed normalized across samples such that each feature has zero mean and unit variance.

complete cross-validation of the tuning parameters with the described 200 test splits could take up to a couple weeks to compute. Luckily, the cross-validation algorithm for performance assessment lends itself nicely to parallelization. By splitting the work across several processors, the computational time can be drastically reduced. For example, in our applications we divided the parameter estimation of the 200 folds amongst eight processors. Thus it took approximately 20 to 45 minutes to obtain a performance estimate ϵ_α for a given value of α. Table 3.5 gives the total computational time via parallel computing required to obtain a cross-validation of the tuning parameters for the classifiers listed in Table 3.4.

3.4 Results

Table 3.4 lists the cross-validation predicted accuracy rates for each of the seven classifiers along with the test accuracy obtained from the 653 day two test samples. Had we participated in the competition, our choice of classifier would have been based on the cross-validation predicted accuracy rates. While all fairly close, the classifier incorporating detrended variance, FPC scores, and network features would have been chosen as our final model as this is one of two classifiers having the highest predicted accuracy rate 61.68% and the fewest number of features of the two. The test accuracy from this classifier is 66.46%, which is just short of 68.0% obtained by the competition winners, but higher than 63.2% accuracy rate obtained by the first runner-up. *Thus with our entry we would have finished in second place.* The confusion matrix for our classifier is presented in Table 3.6. Our classifier has highest accuracy for predicting the 'Chaplin' (92.8%) video clips from the MEG data, and lowest accuracy for predicting the 'Football' (52.9%) video clip.

Table 3.6: Confusion matrix summarizing the performance on the test data for the classifier incorporating detrended variance, FPCA, and network features.

	True stimulus				
Predicted stimulus	Artificial	Nature	Football	Mr. Bean	Chaplin
Artificial	90	27	28	6	3
Nature	39	98	16	6	0
Football	14	12	54	12	4
Mr. Bean	5	11	4	76	2
Chaplin	2	3	0	25	116

In preliminary explorations of the data it was found that including a large number of features in the classifier resulted in a decrease in the cross-validation predicted accuracy rates. While using an elastic net penalty is appealing in situations where there are potentially many more features than samples, as it effectively removes many of the features by setting their estimated coefficients to zero, we think it may be possible to improve the accuracy rates by performing some variable selection prior to learning the classifier as this particular problem seems very sensitive to over-fitting. From this perspective, we briefly introduce block principal component analysis for feature selection from neuroimaging data.

3.4.1 Block Principal Component Analysis

Block principal component analysis (BPCA), as proposed by Liu et al. (2002), is a method of variable selection that can be useful in situations where there are a large number of variables/features and a relatively small number of available samples. Given a large number of features, many of them can be highly correlated with each other and thus some features may become redundant when the rest are being used to explore the data. BPCA seeks to identify and eliminate these redundant features by performing PCA on various subsets of the features, identifying a few significant features within each subset, and then collecting the features selected within the subsets to be passed to the classifier. Given a set of features derived from the training samples, $\boldsymbol{Y}_{lf} = (Y_{lf1}, \ldots, Y_{lfm})', l = 1, \ldots, L$, the BPCA feature selection procedure can be implemented as follows:

Step 1. Group the features into a number of blocks according to their correlation.

Step 2. Perform PCA within each block, selecting as many of the leading principal components as required to explain 95% of the total variation. From these leading principal components, examine the magnitudes of the coefficients of the features in the block and retain only those features with large coefficients.

The features selected from each block are then combined to form the features for classification.

To start, assume we have a set of training samples $\boldsymbol{Y}_l = \{y_{li}(t), i = 1, \ldots, n;\ t = 1, \ldots, T\}$, $l = 1, \ldots, L$, from which we define feature vectors $\boldsymbol{Y}_{l,f} = \{Y_{lf_1}, \ldots, Y_{lf_m})'$.

We begin by grouping the features into blocks according to their correlation, as recommended by Liu et al. (2002). We want the features within a block to have high correlation while the features between two blocks should have low correlation. Thus we group the features according to a hierarchical clustering algorithm where the dissimilarity measure between two features $\boldsymbol{Y}_{fi} = (Y_{1fi}, \ldots, Y_{Lfi})'$ and $\boldsymbol{Y}_{fj} = (Y_{1fj}, \ldots, Y_{Lfj})'$ is taken to be $D(\boldsymbol{Y}_{fi}, \boldsymbol{Y}_{fj}) = 1 - |R(\boldsymbol{Y}_{fi}, \boldsymbol{Y}_{fj})|$, where $R(\boldsymbol{Y}_{fi}, \boldsymbol{Y}_{fj})$ is the Pearson correlation coefficient of the two features. This dissimilarity measure lies in the unit interval and ranks features with high correlation as more similar ($D = 0$) and features with low correlation as less similar ($D = 1$).

We consider an agglomerative clustering algorithm that builds clusters in a 'bottom-up' manner; it begins by allocating each feature to its own cluster then proceeds to iteratively join the two most similar clusters (as defined by some similarity measure) until there is just a single cluster. The clustering procedure can be represented graphically by a dendogram diagram which depicts the feature clusters at all stages of the clustering algorithm. Cutting the dendogram at a specified height defines the feature clusters.

Once the feature blocks have been specified the next step is to determine which features from each block are the most significant, as these will be retained for use in the classifier while the other features will be discarded. BPCA proposes to perform PCA within each block and to associate the features in the block with the principal components (PCs) thus ranking the features by significance. There are several ways to choose which features are associated with the PCs and the number of features associated with each PC; see Jolliffe (1972) for further discussion. For our purposes, we associate one variable with each of the PCs in a given block according to the magnitude of the feature coefficients in the PCs; that is, assuming the PCs are in descending order according to their associated variances, we choose a feature from the current PC with the largest absolute coefficient, provided that feature has not already been chosen from a previous PC. Thus the features within each block are ranked according to the PCs and we choose the first p features from this ordered set, where p is the minimum number of PCs required to explain 95% of the total variability within the block, and discard the remaining features. This procedure is repeated for every block and, afterwards, the features that remain from each block are combined to form the features to be used for classification.

We performed a small experiment with the feature sets used to define the seven classifiers in Table 3.4 to investigate whether reducing the number of features with BPCA prior to training the classifier can improve the accuracy rates for the ICANN dataset. We cluster the features according to a complete linkage agglomeration algorithm in R using the dissimilarity measure based on correlation defined above. Taking the distance measure between clusters to be the complete link $D(C_1, C_2) = \max_{x_1 \in C_1,\, x_2 \in C_2} D(x_1, x_2)$, where x_1, x_2 are elements in the clusters C_1, C_2, respectively, allows for 'similar' clusters to be joined first in the sense that two clusters are joined only if *all* the points in one cluster are close to *all* the points of the other cluster. To choose the number of blocks that the features are to be grouped into we consider a sequence of groupings based on $n = 1, \ldots, 20$ blocks: $\{B_1 = \{B_{1,1}\}, B_2 = \{B_{2,1}, B_{2,2}\}, \ldots, B_{20} = \{B_{20,1}, B_{20,2}, \ldots, B_{20,20}\}\}$. For each block, $B_{i,j}$, $1 \leq j \leq i \leq 20$, we choose $p_{(i,j)}$ features to retain, where $p_{(i,j)}$ is the minimum number of PCs required to explain 95% of the variability in block $B_{i,j}$. For each set of blocks $B_i = \{B_{i,1}, \ldots, B_{i,i}\}$, $i = 1, \ldots, 20$, the features chosen from each of the component blocks $B_{i,j}, j = 1, \ldots, i$, are re-grouped together to form a reduced feature vector $\boldsymbol{Y}_{(B_i)f} = (Y_{(B_i)f_1}, \ldots, Y_{(B_i)f_q})'$.

Of the twenty sets considered, the reduced feature set having the highest predicted accuracy is chosen as the optimal subset from the original features and the optimal number of blocks is taken to be i, where B_i is the associated block set. Table 3.7 gives the results of applying feature selection with BPCA to the seven classifiers considered in Table 3.4. Again, if we had participated in the competition our choice of classifier would have been based on the predicted accuracy rates. Thus, again, we would have chosen the classifier incorporating the detrended variance, FPCA scores, and network features. We can see that by reducing the number of features in this feature set prior to learning the classifier we obtain a higher 66.1% predicted accuracy rate than the 61.68% predicted accuracy previously obtained by using all the features to train the classifier. However, the test accuracy from using this BPCA reduced feature set is only 63.0%, which is less than the 66.46% test accuracy we would have obtained by using the full feature set. Nonetheless, if we were to choose a classifier for submission to the competition we would have chosen the BPCA reduced version of the feature set including detrended variance, FPC scores, and network features as it gives the highest predicted accuracy and we would have still ranked fairly highly, nearly tying with the second place entry (63.2% accuracy rate).

Table 3.7: Results from experiment using BPCA to increase the accuracy of classifiers used for the brain decoding competition dataset. Our previous top classifier (using all features) obtained a CV predicted accuracy of 61.68% and a test accuracy of 66.46%. Note that 'PH' refers to the 12 features derived using persistent homology.

Classifier	# of Blocks	# of Features Retained	CV Predicted Accuracy	Test Accuracy
1 - Detrended variance	19	124	64.8%	58.7%
2 - Detrended variance + FPCA	4	239	63.7%	56.5%
3 - Detrended variance + Network Features	7	107	63.0%	60.6%
4 - Detrended variance + PH	5	106	61.9%	60.8%
5 - Detrended variance + FPCA + Network Features	5	257	66.1%	63.0%
6 - Detrended variance + FPCA + PH	19	319	62.9%	63.6%
7 - Detrended variance + FPCA + Network Features + PH	19	321	62.9%	63.9%

Examining the results of Table 3.7, it is apparent that many of the BPCA reduced feature sets show a significant drop between the predicted and test accuracies. This is likely due to either (i) the BPCA method selecting too few features (we only experimented with 20 different subsets of the original features) or (ii) that the features selected by BPCA do not allow the classifier to generalize well to new data (ie. the classifier is over-learning on the reduced feature sets), thus leading to the poor generalization seen with the test data. Overall, BPCA did not succeed in improving the accuracy of the classifiers for this particular application. Further investigation with a longer sequence of feature blocks may reveal BPCA to be useful for improving the accuracy rates in this application. Furthermore, there is reason to believe BPCA may give better performance in situations where the number of features is significantly larger than the number of training samples, which is not the case for the seven feature sets we considered.

Chapter 4

Conclusions

We have reviewed the brain decoding problem in neuroscience and have discussed approaches from statistics, computational topology, and graph theory for constructing features for this high-dimensional classification problem. We have developed classifiers combining FPCA, persistent homology, and graph theoretic measures derived from mutual information networks. We have considered incorporating the features within a classifier based on symmetric multinomial logistic regression incorporating elastic net regularization and have applied our approach to a real brain decoding competition dataset illustrating good performance.

Overall, examining the results in Table 3.4 we see that those classifiers incorporating FPC scores all perform quite well, with test accuracy scores being higher than predicted accuracy scores. It is not clear to us what aspect of the FPC scores allows for this increase and this requires further investigation. Regarding the global features, there seems to be a small advantage gained in incorporating the network features but nothing gained by incorporating persistent homology. We emphasize that this is only for a single dataset and experimental paradigm. Performance on other brain decoding datasets may yield different results in particular as the samples considered in our application were based on fairly short 1-second recordings. Our limited experimentation with block principal component analysis did not yield any improvement to the accuracies of the classifiers we considered in this application, though we suspect that a more thorough investigation and continued experimentation may show BPCA for variable selection to be useful in situations where there are a large number of neuroimaging features being considered.

There are several avenues for future research in this area; the utility of persistent homology and FPCA for decoding problems involving longer recordings and different experimental paradigms (involving face perception) is one such avenue. Aside from classification, both techniques can also be used to explore and summarize novel aspects of neuroimaging data. Also, given the interesting results we have observed with the classifiers incorporating FPCA, we propose exploring the use of more general approaches based on nonlinear manifold representations for functional data such as those recently proposed by Chen and Müller (2012).

One final area that compels future research is to expand the general methods for brain decoding outlined in this thesis to the multi-subject brain decoding problem, which is much more complex than the single-subject paradigm we considered in Chapter 3. In general, the multi-subject brain decoding problem occurs when there are a group of subjects whose neuroimaging data is used for training and another group of subjects who represent the test set. Having high-dimensional neuroimaging measurements from multiple subjects leads to several complications, the most obvious being that the small domain shift problem encountered in our single-subject application is further compounded by the fact that the test samples come from completely new subjects whose brain signals may not be represented in the training subjects. The issue of extracting information from multiple sources and combining these features to define a classifier presents another obstacle.

Appendix A

The following sections of code were implemented using R version 3.1.1 ("Sock it to Me") and MATLAB version R2012a.

A.1 R code for preliminary data processing

```
## Download Data - read in from matlab file, save as R objects
library(R.matlab)
data=readMat('megicann_train_v2.mat')
save(data, file="megicann_train_v2R")
dataTest=readMat('megicann_test_v2.mat')
save(dataTest, file="megicann_test_v2R")
dataSecret=readMat('megicann_secret.mat')
save(dataSecret, file="megicann_secretR")

## Combine day1 & day2 training class labels into vector [1:727]
train.class=c(data$class.day1,data$class.day2)
save(train.class,file="train_class")

## Combine day1 & day2 training data in array [1:727, 1:204, 1:200]
library(abind)
train.day1_0=array(unlist(data$train.day1[1]),dim=c(677,204,200),
dimnames=c('sample','channel','time'))
train.day2_0=array(unlist(data$train.day2[1]),dim=c(50,204,200),
```

```
dimnames=c('sample','channel','time'))
## Save training data
train_0=abind(train.day1_0,train.day2_0,along=1)
save(train_0, file="train_0")

## Save test data [1:653, 1:204, 1:200]
test_0=array(unlist(dataTest$test.day2[1]),dim=c(653,204,200),
dimnames=c('sample','channel','time'))
save(test_0, file="test_0")

# Calculate detrended timeseries

time=c(1:200)-100.5  ## subtract from 100.5 = midpoint of time indices

intercept0=matrix(NA,nrow=204,ncol=727)
slope0=matrix(NA,nrow=204,ncol=727)
for(k in 1:727){
    for(i in 1:204){
        regr=lm(train_0[k,i,]~time)
        intercept0[i,k]=regr$coefficients[1]
        slope0[i,k]=regr$coefficients[2]
    }
}
## Transpose Training feature matrices to dim [1:727, 1:204]; save
intercept0=t(intercept0); slope0=t(slope0)
save(intercept0, file="intercept0"); save(slope0, file="slope0")

test_intercept0=matrix(NA,nrow=204,ncol=653)
test_slope0=matrix(NA,nrow=204,ncol=653)
for(k in 1:653){
    for(i in 1:204){
        regr=lm(test_0[k,i,]~time)
        test_intercept0[i,k]=regr$coefficients[1]
        test_slope0[i,k]=regr$coefficients[2]
    }
```

```
}
## Transpose Test feature matrices to dim [1:653, 1:204]; save
test_intercept0=t(test_intercept0); test_slope0=t(test_slope0)
save(test_intercept0, file="test_intercept0")
save(test_slope0, file="test_slope0")

train_0.detrend=array(NA,dim=c(727,204,200),
dimnames=c('sample','channel','time'))
for(k in 1:727){
    for(i in 1:204){
        for(n in 1:length(time)){
            train_0.detrend[k,i,n]=
            train_0[k,i,n]-slope0[k,i]*time[n]-intercept0[k,i]
        }
    }
}
save(train_0.detrend, file="train_0_detrend")

test_0.detrend=array(NA,dim=c(653,204,200),
dimnames=c('sample','channel','time'))
for(k in 1:653){
    for(i in 1:204){
        for(n in 1:length(time)){
            test_0.detrend[k,i,n]=
            test_0[k,i,n]-test_slope0[k,i]*time[n]-test_intercept0[k,i]
        }
    }
}
save(test_0.detrend, file="test_0_detrend")

# Calculate Variance ------------------------------------------------

var0.detrend=c()
for(k in 1:727){
    var0.detrend=cbind(var0.detrend,apply(train_0.detrend[k,,],1,var))
```

```
}
## Transpose Training feature matrices to dim [1:727, 1:204]; save
var0.detrend=t(var0.detrend); save(var0.detrend, file="var0_detrend")

test_var0.detrend=c()
for(k in 1:653){
    test_var0.detrend=cbind(test_var0.detrend,
    apply(test_0.detrend[k,,],1,var))
}
## Transpose Test feature matrices to dim [1:653, 1:204]; save
test_var0.detrend=t(test_var0.detrend)
save(test_var0.detrend,file="test_var0_detrend")
```

A.2 R code for FPCA features

```
## Load Required Data
load("train_0", verbose=TRUE); str(train_0)## s.b.[1:727, 1:204, 1:200]
load("test_0", verbose=TRUE); str(test_0)## s.b.[1:653, 1:204, 1:200]

library("fda") ## Require 'fda' package (Ramsay and Silverman, 2005)

# Step 1: Approximate the TS data with smooth curves for each sensor

## Note: this is all the same, does not depend on the data.
timepts=seq(1,200,1)
norder=4 ## cubic B-spline
nbasis=norder+length(timepts)-2; (nbasis) ## 202

## create cubic B-spline basis functions, class 'basisfd'
spline.basis=
    create.bspline.basis(rangeval=c(1,200),nbasis,norder,timepts)

## Approx. curve functions at each obs. for each channel
```

```
SensorChannel=c(1:204) ## s.b. 204 - sensor channel ID

## Fit curves using ALL THE TRAINIG DATA --- Use this.
Data=train_0
FittedCurves=list()

SensorTS=list()
for(i in 1:dim(Data)[2]){
  SensorTS[[i]]=t(Data[,i,])
}
str(SensorTS) # list of 204 matrices size [200, N=727]

## define a functional parameter object that penalizes the roughness
## of fitted curves by using the 2nd derivative in roughness penalty.

for(j in 1:length(SensorChannel)){
  cat(paste("Fitting curves for sensor channel =",j,"\n"))
  fdPar=fdPar(spline.basis, 2, 10^-2) ## info on smoothing
  sensor.fit=smooth.basis(timepts, SensorTS[[j]], fdPar) ## data
  sensor.fd=sensor.fit$fd ## functional data object
  sensor.fd$fdnames=list("Time (seconds/200)",
                         "Sample Number"=NULL,"MEG Measurement")
  FittedCurves[[j]]=sensor.fd
}

# Step 2: Extract FPCA features for each sensor channel

## Define params for smoothed FPCs
D2Lfd=int2Lfd(m=2)
D2fdPar=fdPar(spline.basis, D2Lfd, 1e4)
N_HARM=4  ## number of harmonics/FPCs to compute

## Get FPC functions/weights based on ALL TRAINING DATA

## Construct smooth pca.fd object for each channel's smoothed curves
```

```
PCAobjects.smooth=lapply(FittedCurves, pca.fd,
                                    nharm=N_HARM, harmfdPar=D2fdPar)

# FVE by each PC for each channel; list of 204 num[1,m=N_HARM]
FVE.smooth=lapply(PCAobjects.smooth, function(X) X$varprop);FVE.smooth

FVE.FPC1.smooth=sapply(FVE.smooth, function(X) X[1])
FVE.FPC2.smooth=sapply(FVE.smooth, function(X) X[2])
FVE.FPC3.smooth=sapply(FVE.smooth, function(X) X[3])
FVE.FPC4.smooth=sapply(FVE.smooth, function(X) X[4])

## Use enough FPCs to get 90% FVE at each sensor
FVE_90=lapply(PCAobjects.smooth, function(X) X$varprop); FVE_90
length(which(sapply(FVE_90,sum)<.90)); which(sapply(FVE_90,sum)<.90)
## number of FPCs needed at each sensor to have 90 FVE
num_FPCs=rep(1,204)
cum_FVE=sapply(FVE_90, function(X) X[1]) ## First FPC FVE
for(i in 1:204){
  while(cum_FVE[i]<.90) {
    num_FPCs[i]=num_FPCs[i]+1
    cum_FVE[i]=cum_FVE[i]+FVE_90[[i]][num_FPCs[i]]
  }
}
sum(num_FPCs) ## 452

## Define and save FPC score features

# Use ALL the Training data to define weights
# using min. num of FPCs at each sensor to have 90% FVE
scores.combined_90.smooth=list() ## use all N_HARM FPCs
for(i in 1:204){
  ## Weight w(t) based on all training
  weight=PCAobjects.smooth[[i]]$harmonics
  ## Subt. mean curve from all training
  mean.curve=mean(FittedCurves[[i]])
```

```
  a=mean.curve
  m=matrix(a$coefs,nbasis,727)
  a$coefs=m
  ## Calc. score for all training samples
  c=matrix(inprod(weight, FittedCurves[[i]]-a),
               ncol=N_HARM, byrow=TRUE)

  scores.combined_90.smooth[[i]]=c
}

## extract necessary scores for each sensor based on num_FPCs
scores_90.smooth=list()
for(i in 1:204){
  scores_90.smooth[[i]]=scores.combined_90.smooth[[i]][,1:num_FPCs[i]]
}

Scores_90_features.smooth=matrix(unlist(scores_90.smooth), nrow=727)
save(Scores_90_features.smooth, file='scores_90_smooth_train')

scores.combined_90.Test.smooth=list()
for(i in 1:204){
  ## Weight w(t) based on all training
  weight=PCAobjects.smooth[[i]]$harmonics
  ## Subt. mean curve from testing samples
  mean.curve=mean(FittedCurves.Test[[i]])
  a=mean.curve
  m=matrix(a$coefs,nbasis,653)
  a$coefs=m
  ## Calc. score for all testing samples
  c=matrix(inprod(weight, FittedCurves.Test[[i]]-a),
               ncol=N_HARM, byrow=TRUE)

  scores.combined_90.Test.smooth[[i]]=c
}
```

```
## extract necessary scores for each sensor based on num_FPCs
scores_90.Test.smooth=list()
for(i in 1:204){
  scores_90.Test.smooth[[i]]=
        scores.combined_90.Test.smooth[[i]][,1:num_FPCs[i]]
}

Scores_90_features.Test.smooth=
        matrix(unlist(scores_90.Test.smooth), nrow=653)
save(Scores_90_features.Test.smooth, file='scores_90_smooth_test')
```

A.3 MATLAB code for mutual information network features

```
%% Requires the Functional Connectivity Toolbox (Zhou et. al, 2009)
%% Requires the Brain Connectivity Toolbox (Rubinov and Sporns, 2010)

% Step 1 - Calculate Mutual Information matrices  for training data

%% Load the data and extract the freq0 samples,
%% combine into one array hz0 <727x204x200>
load('megicann_train_v2.mat')
hz0_day1=train_day1{1}; %% Extract the raw MEG signals
hz0_day2=train_day2{1}; %% Extract the raw MEG signals
hz0=[hz0_day1; hz0_day2];

% Break up into smaller groups of samples to speedup calculations
%(parallelization)
hz1=hz0(1:72,:,:); size(hz1)
hz2=hz0(73:144,:,:); size(hz2)
hz3=hz0(145:216,:,:); size(hz3)
hz4=hz0(217:288,:,:); size(hz4)
hz5=hz0(289:360,:,:); size(hz5)
```

```
hz6=hz0(361:432,:,:); size(hz6)
hz7=hz0(433:504,:,:); size(hz7)
hz8=hz0(505:576,:,:); size(hz8)
hz9=hz0(577:648,:,:); size(hz9)
hz10=hz0(649:727,:,:); size(hz10)

%%%%%%%%%%
%% The following section of code is to be run for each of hz1:hz10
%% making appropriate substitutions for Si where i=1:10
%%%%%%%%%%

WHICH=hz1; %% Determines which subset of samples to use

a=size(WHICH); %% N 204 200
N=a(1);

%create an empty 1xN cell array to hold the reshaped TS matrices
T=cell(1,N);

for k=1:N
    %% need TS matrices to be column-wise (ie. 200x204)
    T{k}=transpose(reshape(WHICH(k,:,:),[204,200]));
end

%create an empty 1xN cell array to hold the 'mutualinf' structs
S1=cell(1,N);

%% For each sample calculate the Mutual information matrix
for k=1:N
    %% need TS matrices to be column-wise (ie. 200x204)
    y=T{k};
    [n,m]=size(y);
    % mututal information
    s.phi=eye(m);
    for i=1:(m-1)
```

```
      for j= (i+1):m
        s.phi(i,j)=mutualinf(y(:,i),y(:,j));
        s.phi(j,i)=s.phi(i,j);
      end
    end
    S1{k}=s.phi;
end

%% rename file name with appropriate index (eg. 1:10)
save MIConnectivityMatrices1_Train S1

%%%%%%%%%% End parallelization

%% Following successful compilation of above section, should have
%% objects: S1,S2,...,S9,S10

% Append all cells arrays to get <1x727> cell array
Sorig=[S1,S2,S3,S4,S5,S6,S7,S8,S9,S10];
a=size(Sorig); %% 1x727 cell; each cell contains MI matrix 204x204
N=a(2) %% 727 = number of samples

% Remove all self-connections
S=cell(1,N);  %create an empty 1xN cell array
for k=1:N
[n,m]=size(Sorig{k});
S{k}=Sorig{k}-eye(m);
end

%% set proportion of correlation entries to keep
p=0.2;
THRESH=cellfun(@(x) threshold_proportional(x,p), S,
                                          'UniformOutput', false);

% Step 2 - Compute the network measures from mutual information matrix
```

```
%% Clustering coefficient
ClusteringCoef=cellfun(@(x) clustering_coef_wu(x),
                                    THRESH, 'UniformOutput', false);
ClusterCoef=transpose(cell2mat(ClusteringCoef));
save MI_ClusterCoef ClusterCoef;

%% Transitivity
Transitivity=cell2mat(transpose(cellfun(@(x) transitivity_wu(x),
                                    THRESH, 'UniformOutput', false)));
save MI_Transitivity Transitivity;

%% Local Efficiency
LocalEfficiency=cellfun(@(x) efficiency_wei(x,1),
                                    THRESH, 'UniformOutput', false);
LocalEfficiency=transpose(cell2mat(LocalEfficiency));
save MI_LocalEfficiency LocalEfficiency;

%% Global Efficiency
GlobalEfficiency=cellfun(@(x) efficiency_wei(x,0),
                                    THRESH, 'UniformOutput', false);
GlobalEfficiency=transpose(cell2mat(GlobalEfficiency));
save MI_GlobalEfficiency GlobalEfficiency;

%% Assortativity Coefficient
Assortativity=cell2mat(transpose(cellfun(@(x) assortativity_wei(x,0),
                                    THRESH, 'UniformOutput', false)));
save MI_Assortativity Assortativity;

%% Modularity
% NOTE : Modularity changes with different runs
[CommunityStructure,Modularity]=
    cellfun(@(x) modularity_louvain_und(x,1), THRESH,
                                          'UniformOutput', false);
Modularity=cell2mat(transpose(Modularity));
save  MI_Modularity Modularity;
```

```
%% Distance and Characteristic Path Length

% The distance matrix contains lengths of shortest paths between all
% pairs of nodes.
% Get Distance matrices
Length=cellfun(@(x) power(x,-1), THRESH, 'UniformOutput', false);
Distance=cellfun(@(x) distance_wei(x), Length, 'UniformOutput', false);
save MI_Distance Distance;

% Calculate Characteristic Path Length
CharPathLength=cell2mat(transpose(cellfun(@(x)
        sum(sum(x(x~=Inf)))/length(nonzeros(x~=Inf)), Distance,
                                            'UniformOutput', false)));
save MI_CharPathLength CharPathLength;
```

A.4 R code for persistent homology features

```
## Requires 'TDA' package (Fasy et. al, 2004)

# Load training data
library(TDA)
load("train_0",verbose=TRUE) ## str [1:727, 1:204, 1:200]

# Standardize timeseries at each sensor (zero mean and unit variance)
standardize=function(point.cloud){
  t=dim(point.cloud)[2]
  m=apply(point.cloud,1,mean)
  s=apply(point.cloud,1,sd)
  return((point.cloud-m)/s)
}
## standardized data = train_0.std
train_0.std=array(NA, dim=c(727,204,200))
for(i in 1:727){
```

```
  train_0.std[i,,]=standardize(train_0[i,,])
}

# Distance metric
abs.corr.dist=function(m){
  return(1-m^2)
}
## cor gives pairwise correlations of columns of a matrix
## need to transpose point cloud, so points are columns
pc=train_0.std[1,,]
p=cor(t(pc)) ## str(p) should be 204x204

library(plyr)
## list of 727 distance matrices 204X204
distances=alply(train_0.std,1,function(x) abs.corr.dist(cor(t(x))))

# Compute Persistent Homology

max_dim=2  ## Max Dimension for betti_numbers (0,1,2)
max_f=0.975 ## Max filtration value

############
## Expedite calculation of filtration sequence by running the
## following section for each group of samples, group in {1,2,...,11}
############

sample.group=list(1:68, 69:136, 137:204, 205:272, 273:340, 341:408,
                        409:476, 477:544, 545:612, 613:680, 681:727)
sapply(sample.group,length)

## which group samples belong to {1,2,...,11}; change as needed
GROUP=1

Rips=lapply(distances[sample.group[[GROUP]]], ripsDiag,
                                maxdimension=max_dim, maxscale=max_f,
```

```
                                dist="arbitrary", printStatus=TRUE)

## Replace all Inf death times with max_f*2
RipsGroup1=lapply(Rips,function(X)
                             replace(X,which(is.infinite(X)),max_f*2))

## Save list of 'diagrams' objects
## change object and filename to reflect group number {1,2,...,11}
save(RipsGroup1, file="RipsGroup1")

############
## Upon successful compilation of above section, should have objects
## containing the filtrations of the training data:
## RipsGroup1, RipsGroup2, ..., RipsGroup11
############

# Object Definitions

## RipsTrain:  Combine all grouped 'diagram' objects into single list
## of 'diagrams' for each sample in training set [1:727]
RipsTrain=c(RipsGroup1,RipsGroup2,RipsGroup3,RipsGroup4,RipsGroup5,
  RipsGroup6,RipsGroup7,RipsGroup8,RipsGroup9,RipsGroup10,RipsGroup11)

## Persistence :
PH.train=lapply(RipsTrain, function(X)
                        cbind(X,Persistence=X[,'Death']-X[,'Birth']))

## Sort training data by Dimension {0,1,2}
Dim0Train=lapply(PH.train, function(X)
                        X[which(X[,'dimension']==0),'Persistence'])
Dim1Train=lapply(PH.train, function(X)
                        X[which(X[,'dimension']==1),'Persistence'])
Dim2Train=lapply(PH.train, function(X)
                        X[which(X[,'dimension']==2),'Persistence'])
str(Dim0Train); str(Dim1Train); str(Dim2Train)
```

```
# Feature Definitions

# Feature: Persistence Size
PS_0_Train=sapply(Dim0Train, sum)/2
PS_1_Train=sapply(Dim1Train, sum)/2
PS_2_Train=sapply(Dim2Train, sum)/2

# Feature: Variance of Persistence
PV_0_Train=sapply(Dim0Train, var)
PV_1_Train=sapply(Dim1Train, var)
PV_2_Train=sapply(Dim2Train, var)

library(moments)
# Feature: Skewness of Persistence
PSk_0_Train=sapply(Dim0Train, skewness)
PSk_1_Train=sapply(Dim1Train, skewness)
PSk_2_Train=sapply(Dim2Train, skewness)

# Feature: Kurtosis of Persistence
PK_0_Train=sapply(Dim0Train, kurtosis)
PK_1_Train=sapply(Dim1Train, kurtosis)
PK_2_Train=sapply(Dim2Train, kurtosis)
```

A.5 R code for error estimation of classifier

```
## NOTE:  Algorithm for cross-validation of tuning parameters
## (Alpha, Lambda) with 200-folds, gives error estimate of classifier
## incorporating features from:
## Detrended Variance && FPCA Scores && Global Network Features

sink("NestedCV_for_DetrendedVariance_FPCA_MINetwork.txt",split=TRUE)

# Load data for feature set ----------------------------------------
```

```
load("train_class") ## class labels for training data

## Load Detrended Variance
load("var0_detrend", verbose=TRUE); str(var0.detrend) #[1:727, 1:204]

## Scores_90_features.smooth matrix [1:727, 1:'n'] FPC scores based on
## varying number of FPCS for the TRAINING data based on combined w(t)
load(file="scores_90_smooth_train", verbose=TRUE)
str(Scores_90_features.smooth)

## Load Global network analysis features based on MI weighted network
load("MI_CharPathLength_R", verbose=TRUE)
str(MI_CharPathLength) # num [1:727, 1]
load("MI_GlobalEfficiency_R", verbose=TRUE)
str(MI_GlobalEfficiency) # num [1:727, 1]
load("MI_NetworkClusterCoef_R", verbose=TRUE)
MI_NetworkClusterCoef=matrix(MI_NetworkClusterCoef,ncol=1)
str(MI_NetworkClusterCoef)# num [1:727,1]
load("MI_Transitivity_R", verbose=TRUE)
str(MI_Transitivity) # num [1:727, 1]
load("MI_NetworkLocalEfficiency_R", verbose=TRUE)
str(MI_NetworkLocalEfficiency) # num [1:727]
load("MI_Modularity_R", verbose=TRUE)
str(MI_Modularity) # num [1:727, 1]
load("MI_Assortativity_R", verbose=TRUE)
str(MI_Assortativity) # num [1:727, 1]

features=list("DetrendedVariance & FPCScores & GlobalNetworkFeatures")

# Global Variables ----------------------------------------------------
library(parallel) ## requires 'parallel' package
library(glmnet) ## requires 'glmnet' package
glmnet.control(mxit=200)
```

```
Alpha.Seq=seq(0,1,0.1)
names(Alpha.Seq)=seq(0,1,0.1)
Lambdas=exp(seq(-8,0,length.out=200))
N=200
M=5

SecondDayWeight=3*50/(677+3*50)
CV.SecondDayWeight=3*25/(677+3*25)

DAY1=c(1:677)
DAY2=c(678:727)

WEIGHTS=c()
WEIGHTS[DAY1]=(1-SecondDayWeight)/677
WEIGHTS[DAY2]=SecondDayWeight/50
#sum(WEIGHTS)==1

CV.WEIGHTS=c()
CV.WEIGHTS[DAY1]=(1-CV.SecondDayWeight)/677
CV.WEIGHTS[DAY2]=CV.SecondDayWeight/25
CV.WEIGHTS=CV.WEIGHTS[1:702]
#sum(CV.WEIGHTS)==1

# Create the fold index for N=200 folds
# NOTE: The same foldIndex should be used for all ALPHA values
set.seed(1)  # for reproducibility  (still variability in inner CV loop)
foldIndex=matrix(NA,ncol=50)  ## matrix [200, 50] {1=TEST, 0=TRAIN}
x=c(rep(0,25),rep(1,25))
foldIndex[1,]=sample(x)  # initiate matrix
while(dim(foldIndex)[1]<N) {
  s.temp=sample(x)
  checkDup=any(apply(foldIndex,1,function(x,want)
                                    isTRUE(all.equal(x,want)),s.temp))
  if(checkDup==TRUE){} #do nothing
  else foldIndex=rbind(foldIndex,s.temp)
```

```
}
rownames(foldIndex)=NULL

# Global Functions ------------------------------------------------
standardize=function(feat.matrix,mx,sx){
  n=dim(feat.matrix)[1]
  m=matrix(rep(mx,n),nrow=n,byrow=TRUE)
  s=matrix(rep(sx,n),nrow=n,byrow=TRUE)
  return((feat.matrix-m)/s)
}
mX=function(feat.matrix){
  return(apply(feat.matrix, MARGIN = 2, mean))
}
sX=function(feat.matrix){
  return(apply(feat.matrix, MARGIN = 2, sd))
}

FeatureSetError=function(XTrain,ALPHA=NULL){
  retval=list()  ## list to keep track of important info to return
  st=Sys.time()

  # Standardize Input Feature Matrix --------------------------------
  p=dim(XTrain)[2]
  retval$p=p

  XTrain=standardize(XTrain,mX(XTrain),sX(XTrain))
  XTrain.day1=XTrain[DAY1,]   #[1:677, 1:p]
  XTrain.day2=XTrain[DAY2,]  #[1:50, 1:p]

  YTrain=as.factor(train.class)
  YTrain.day1=YTrain[DAY1]
  YTrain.day2=YTrain[DAY2]

  # Functions (dependent on XTrain=FeatureSet) ----------------------
  GenerateLambdaSeq=function(Alpha){
```

```
  lambda.seq=glmnet(XTrain,YTrain,weights=WEIGHTS,alpha=Alpha,
                    family='multinomial')$lambda
  return(lambda.seq)
}

Algorithm1=function(ALPHA,FOLDID=NULL){
  retval$Alpha=ALPHA
  retval$N=N

  Alg1.st=Sys.time()

  a=which(Alpha.Seq==ALPHA)
  w=which(Lambdas>=range(Alpha.Lambdas[a])[1]&
                                Lambdas<=range(Alpha.Lambdas[a])[2])
  lambda.seq=Lambdas[w]

  retval$whichLambdas=w

  if(length(lambda.seq)==0) {
    retval$LambdaSeqError="Error with Alpha/Lambda seq or Lambda"
  } else {
    ## inner.cv cross validates for parameter lambda
    inner.cv=function(fold){
      ## fold is int 1:200 indicating which outer cv fold
      valid=which(foldIndex[fold,]==0)
      test=which(foldIndex[fold,]==1)
      Training=XTrain.day1  ## associated YTrain.day1
      Validation=XTrain.day2[valid,]
      ValidationClass=YTrain.day2[valid]

      modelXTrain=rbind(Training,Validation)
      modelYTrain=c(YTrain.day1,ValidationClass)

      ## check if FOLDID exists, ie. cross-validation alpha too
      if(!(is.null(FOLDID))){
```

```
    fit=cv.glmnet(modelXTrain,modelYTrain,nfolds=M,alpha=ALPHA,
                  family="multinomial",type.measure="class",
                  lambda=lambda.seq,weights=CV.WEIGHTS)
    retval$time=Sys.time()-Alg1.st
    return(fit)
  } else {#do nothing}
}

## store inner.cv results for each outer fold
glmnet.objects=mclapply(c(1:N),inner.cv,mc.cores=8)

min.nlambda=min(sapply(lapply(glmnet.objects,'[[','cvm'),length))

cv.error.fun=function(fold){
  test=which(foldIndex[fold,]==1)
  Test=XTrain.day2[test,]; TestClass=YTrain.day2[test]

  modelXTest=Test; modelYTest=TestClass

  fit=glmnet.objects[[fold]]
  modelYPredict=predict(fit,newx=modelXTest,s=fit$lambda,
                                                  type='class')

  errors=apply(modelYPredict,2,
                        function(x) table(x==modelYTest)[1]/25)
  return(errors) ## [1:nlambda]
}

# error.alpha.n = list of N [1:nlambdas, errors]
error.alpha.n=lapply(c(1:N),cv.error.fun)
m=matrix(NA,nrow=N,ncol=min.nlambda)  ## dim [1:N, 1:max.nlambda]
for(i in 1:N){
  m[i,]=as.vector(error.alpha.n[[i]])[1:min.nlambda]
}  ## colMeans give test error for given lambda
retval$error=colMeans(m)
```

```
    training.error.fun=function(fold){
      Training=XTrain.day1  ## associated YTrain.day1

      fit=glmnet.objects[[fold]]
      modelTPredict=predict(fit,newx=Training,s=fit$lambda,
                                                    type='class')
      training.errors=apply(modelTPredict,2,
                          function(x) table(x==YTrain.day1)[1]/677)
      return(training.errors) ## [1:nlambda]
    }
    # error.train.n = list of N [1:nlambdas, errors]
    error.train.n=lapply(c(1:N),training.error.fun)
    m=matrix(NA,nrow=N,ncol=min.nlambda) ## [1:N, 1:max.nlambda]
    for(i in 1:N){
      m[i,]=as.vector(error.train.n[[i]])[1:min.nlambda]
    }  ## colMeans give test error for given lambda
    retval$train.error=colMeans(m)

    retval$time=Sys.time()-Alg1.st
  }

  return(retval) ## returns a list
}

# Estimate alpha-errors for feature set ----------------------------
if(is.null(ALPHA)){
  ## (2) Error Estimate via Cross-Validation of Alpha

  # Get lambda sequence for each Alpha
  # list of 11, for each Alpha, num[1:(nlambda, variable)]
  Alpha.Lambdas=mclapply(Alpha.Seq,FUN=GenerateLambdaSeq,mc.cores=4)

  ## NOTE cv.glmnet: If users would like to cross-validate alpha as
  ## well, they should call cv.glmnet with a pre-computed vector
```

```
    ## foldid, and then use this same fold vector in separate calls to
    ## cv.glmnet with different values of alpha
    set.seed(80) # for reproducibility
    x=rep(c(1:M),length.out=702)
    ## foldID = dim [N, 702] matrix of c(1:5)
    foldID=matrix(x,ncol=702,byrow=TRUE)
    while(dim(foldID)[1]<N){
      foldID=rbind(foldID,sample(x))
    }

    ## Run for all alphas in {0,0.1,...,1}
    error.alpha=list(NULL)
    for(i in 1:length(Alpha.Seq)){
      a=Alpha.Seq[i]
      print(paste("ALPHA=",a))
      error.alpha[[i]]=Algorithm1(a,foldID)  ## returns list
    }
    names(error.alpha)=Alpha.Seq
    error.alpha$TotalTime=Sys.time()-st
    print(Sys.time()-st)
    return(error.alpha)
  }else{# do nothing}
}

# Execution begins ------------------------------------------------

print(features[[1]])
Feature_Matrix=cbind(var0.detrend,Scores_90_features.smooth,
                     MI_CharPathLength,MI_GlobalEfficiency,
                     MI_NetworkClusterCoef,MI_Transitivity,
                     MI_NetworkLocalEfficiency,
                     MI_Modularity,MI_Assortativity)
FSE_DetrendedVariance_AND_90FVE_AND_GlobalWeightedMI=
                                        FeatureSetError(Feature_Matrix)
print(str(FSE_DetrendedVariance_AND_90FVE_AND_GlobalWeightedMI))
```

```
save(FSE_DetrendedVariance_AND_90FVE_AND_GlobalWeightedMI,
     file="NestedCV_DetrendedVariance_AND_90FVE_AND_GlobalWeightedMI")
```

A.6 R code for test accuracy of classifier

```
## NOTE: Classifier incorporating features from:
## Detrended Variance && FPCA Scores && Global Network Features

# Global Variables
Alpha.Seq=round(seq(0,1,0.1),1); names(Alpha.Seq)=seq(0,1,0.1)
Lambdas=exp(seq(-8,0,length.out=200))
SecondDayWeight=3*50/(677+3*50)
DAY1=c(1:677); DAY2=c(678:727)
WEIGHTS=c()
WEIGHTS[DAY1]=(1-SecondDayWeight)/677
WEIGHTS[DAY2]=SecondDayWeight/50
M=5 ## number of folds for lambda CV

# Functions
standardize=function(feat.matrix,mx,sx){
    n=dim(feat.matrix)[1]
    m=matrix(rep(mx,n),nrow=n,byrow=TRUE)
    s=matrix(rep(sx,n),nrow=n,byrow=TRUE)
    return((feat.matrix-m)/s)
}
mX=function(feat.matrix){
    return(apply(feat.matrix, MARGIN = 2, mean))
}
sX=function(feat.matrix){
    return(apply(feat.matrix, MARGIN = 2, sd))
}

# Load data for feature set
load("train_class",verbose=TRUE)
```

```
load("megicann_secretR",verbose=TRUE) #str(dataSecret)
## Classes of secret test data, released after competition ended
YTrue=dataSecret$class.test.day2
detailedTest=dataSecret$detailedTest
detailedTrain=dataSecret$detailedTrain
detailedValid=dataSecret$detailedValid

## Load Detrended Variance features
load("var0_detrend",verbose=TRUE); ## var0.detrend
load("test_var0_detrend",verbose=TRUE); ##  test_var0.detrend

## Load FPC scores 90FVE Features
## Scores_90_features.smooth
load("scores_90_smooth_train",verbose=TRUE);
##  Scores_90_features.Test.smooth
load("scores_90_smooth_test",verbose=TRUE);

## Load Mutual Information Network Features
## s.b. in form train [1:727, 1], test [1:653, 1]

load("MI_CharPathLength_R", verbose=TRUE)
load("MI_GlobalEfficiency_R", verbose=TRUE)
load("MI_NetworkClusterCoef_R", verbose=TRUE)
MI_NetworkClusterCoef=matrix(MI_NetworkClusterCoef,ncol=1)
load("MI_Transitivity_R", verbose=TRUE)
load("MI_NetworkLocalEfficiency_R", verbose=TRUE)
load("MI_Modularity_R", verbose=TRUE)
load("MI_Assortativity_R", verbose=TRUE)

load("MI_CharPathLength_Test_R", verbose=TRUE)
load("MI_GlobalEfficiency_Test_R", verbose=TRUE)
load("MI_NetworkClusterCoef_Test_R", verbose=TRUE)
MI_NetworkClusterCoef_Test=matrix(MI_NetworkClusterCoef_Test,ncol=1)
load("MI_Transitivity_Test_R", verbose=TRUE)
load("MI_NetworkLocalEfficiency_Test_R", verbose=TRUE)
```

```
load("MI_Modularity_Test_R", verbose=TRUE)
load("MI_Assortativity_Test_R", verbose=TRUE)

## We consider the classifier including :
## Detrended Variance && FPCA Scores && Global Network Features
# XTrain=cbind(var0.detrend,Scores_90_features.smooth,
#              MI_CharPathLength,MI_GlobalEfficiency,
#              MI_NetworkClusterCoef,MI_Transitivity,
#              MI_NetworkLocalEfficiency,
#              MI_Modularity,MI_Assortativity)
# XTest=cbind(test_var0.detrend,Scores_90_features.Test.smooth,
#             MI_CharPathLength_Test,
#             MI_GlobalEfficiency_Test,MI_NetworkClusterCoef_Test,
#             MI_Transitivity_Test,MI_NetworkLocalEfficiency_Test,
#             MI_Modularity_Test,MI_Assortativity_Test)


############
# Predict Training Set
############
## load results of CV alpha (nested 200-CV scheme)
## FSE_DetrendedVariance_AND_90FVE_AND_GlobalWeightedMI
load("NestedCV_DetrendedVariance_AND_90FVE_AND_GlobalWeightedMI",
verbose=TRUE)
dat=FSE_DetrendedVariance_AND_90FVE_AND_GlobalWeightedMI
##  The range of predicted accuracy for each Alpha value
for(i in 1:11){
    print(range(1-dat[[i]]$error))
}

z=matrix(NA,nrow=length(Lambdas),ncol=length(Alpha.Seq))
max.lambdas=c(); min.lambdas=c()
for(i in 1:11){
    d=dat[[i]]  ## lambdas
    acc=1-d$error; acc=rev(acc)  ## Lambdas s.b. in decreasing order
```

```
    print(length(d$whichLambdas))
    z[d$whichLambdas[1:length(acc)],i]=acc
    max.lambdas[i]=max(acc); min.lambdas[i]=min(acc)
    m=min(d$whichLambdas)
}

## Worst, Best accuracy for all Alphas
min(max.lambdas); max(max.lambdas) ## 0.5814, 0.6168
(max(max.lambdas)-min(max.lambdas))*100  ## 3.54

names(max.lambdas)=Alpha.Seq; max.lambdas
## Choose which Alpha tuning parameter value to use.
which(max.lambdas==max(max.lambdas)) ## Alpha=1, cv performance=0.6168

## Train classifier with chosen Alpha, using 5-fold CV for lambda
## (this time using all training samples)

# Predict Secret Test Set ------------------------------------------
ALPHA=1.0  ## Optimal Alpha as chosen by CV error surface

XTrain=cbind(var0.detrend,Scores_90_features.smooth,MI_CharPathLength,
MI_GlobalEfficiency,MI_NetworkClusterCoef,
MI_Transitivity,MI_NetworkLocalEfficiency,
MI_Modularity,MI_Assortativity)
XTest=cbind(test_var0.detrend,Scores_90_features.Test.smooth,
MI_CharPathLength_Test,MI_GlobalEfficiency_Test,
MI_NetworkClusterCoef_Test,MI_Transitivity_Test,
MI_NetworkLocalEfficiency_Test,MI_Modularity_Test,
MI_Assortativity_Test)
p=dim(XTrain)[2]; p

# Process training/testing feature set -----------------------------

m=mX(XTrain) #[1:p]
s=sX(XTrain) #[1:p]
```

```
XTrain=standardize(XTrain,m,s)
XTrain.day1=XTrain[DAY1,]   #[1:677, 1:p]
XTrain.day2=XTrain[DAY2,]  #[1:50, 1:p]

YTrain=as.factor(train.class)
YTrain.day1=YTrain[DAY1]
YTrain.day2=YTrain[DAY2]

XTest=standardize(XTest,m,s) #[1:653,1:p]

# Select best lambda 5-fold CV
library(glmnet)
glmnet.control(mxit=200)

set.seed(23) # for reproducibility
cv.fit=cv.glmnet(XTrain,YTrain,nfolds=M,alpha=ALPHA,
family="multinomial",type.measure="class",
weights=WEIGHTS)

error.lambda=cv.fit$cvm
LAMBDA=max(cv.fit$lambda[which(error.lambda==min(error.lambda))])
LAMBDA; log(LAMBDA) ## 0.006440608, -5.045132
lambda.ind=which(cv.fit$lambda==LAMBDA)

# Predict classes of secret data based on best lambda model
YPredict=predict(cv.fit,XTest,s=LAMBDA,type='class') ## [1:653,1]
YPredict=as.factor(YPredict)

# How well does our prediction do?

# Accuracy: The ratio of correct predictions
accuracy=length(which(YPredict==YTrue))/length(YTrue)
accuracy ## 0.6646248
```

```
# Confusion Matrix: Predictions as rows, True classes as columns
conf.mat=table("predicted class"=YPredict,"true class"=YTrue)
conf.mat

# Clip Versus Plot: Accuracy in the binary task of separating the
# short clips from the films with a plot
clipVplot=(sum(conf.mat[1:3,1:3])+sum(conf.mat[4:5,4:5]))/653
clipVplot #0.914242

# WithinClip: Accuracy for recognizing the short clips
ind=which(YTrue<4 & as.numeric(YPredict)<4)
withinClip=length(which(YTrue[ind]==YPredict[ind]))/length(ind)
withinClip #0.6402116

# WithinPlot: Accuracy for recognizing the films
ind=which(YTrue>3 & as.numeric(YPredict)>3)
withinPlot=length(which(YTrue[ind]==YPredict[ind]))/length(ind)
withinPlot #0.8767123

# In Tranining Set: Accuracy for samples from stimulus shown also
# during training
ind=which(detailedTest[,3]==1)
inTrain=length(which(YTrue[ind]==YPredict[ind]))/length(ind)
inTrain #0.6804598

# Not in Training Set: Accuracy for new stimuli
ind=which(detailedTest[,3]==0)
notInTrain=length(which(YTrue[ind]==YPredict[ind]))/length(ind)
notInTrain #0.6330275
```

Bibliography

[1] Aaron Adcock, Daniel Rubin, and Gunnar Carlsson. Classification of hepatic lesions using the matching metric. *Computer Vision and Image Understanding*, 121:36–42, 2014.

[2] Peter Bubenik. Statistical topological data analysis using persistence landscapes. *Journal of Machine Learning Research*, 16:77–102, 2015.

[3] Gunnar Carlsson. Topology and data. *Bulletin of the American Mathematical Society*, 46(2):255–308, 2009.

[4] Olivier Chapelle, Patrick Haffner, and Vladmir N. Vapnik. Support vector machines for histogram-based image classification. *IEEE Transactions on Neural Networks*, 10(5):1055–1064, 1999.

[5] Dong Chen and Hans-Georg Müller. Nonlinear manifold representations for functional data. *Annals of Statistics*, 40(1):1–29, 2012.

[6] Moo K. Chung, Peter Bubenik, and Peter T. Kim. Persistence diagrams of cortical surface data. In *Information Processing in Medical Imaging*, pages 386–397. Springer Berlin Heidelberg, 2009.

[7] Brittany T. Fasy, Jisu Kim, Fabrizio Lecci, and Clément Maria. Introduction to the R package TDA. arXiv preprint arXiv:1411.1830, 2014.

[8] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010.

[9] Karl Friston, Carlton Chu, Janaina Mourão-Miranda, Oliver Hulme, Geraint Rees, Will Penny, and John Ashburner. Bayesian decoding of brain images. *NeuroImage*, 39(1):181–205, 2008.

[10] Matti Hämäläinen, Riitta Hari, Risto J. Ilmoniemi, Jukka Knuutila, and Olli V. Lounasmaa. Magnetoencephalography: theory, instrumentation, and applications to noninvasive studies of the working human brain. *Reviews of Modern Physics*, 65(2):413–497, 1993.

[11] John-Dylan Haynes and Geraint Rees. Decoding mental states from brain activity in humans. *Nature Reviews Neuroscience*, 7(7):523–534, 2006.

[12] Giseon Heo, Peter Kim, and Jennifer Gamble. Topological analysis of variance and the maxillary complex. *Journal of the American Statistical Association*, 107(498):477–492, 2012.

[13] Heikki Huttunen, Tapio Manninen, Jukka-Pekka Kauppi, and Jussi Tohka. Mind reading with regularized multinomial logistic regression. *Machine Vision and Applications*, 24(6):1311–1325, 2013.

[14] Harry Joe. Relative entropy measures of multivariate dependence. *Journal of the American Statistical Association*, 84(405):157–164, 1989.

[15] Ian T. Jolliffe. Discarding variables in a principal component analysis. I: Artificial data. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 21(2):160–173, 1972.

[16] Arto Klami, P. Ramkumar, S. Virtanen, L. Parkkonen, R. Hari, and S. Kaski. ICANN/PASCAL2 challenge: MEG mind reading — overview and results. In *Proceedings of ICANN/PASCAL2 Challenge: MEG Mind Reading*, 2011.

[17] Xiaoyan Leng and Hans-Georg Müller. Classification using functional data analysis for temporal gene expression data. *Bioinformatics*, 22(1):68–76, 2006.

[18] Aiyi Liu, Ying Zhang, Edmund Gehan, and Robert Clarke. Block principal component analysis with application to gene microarray data classification. *Statistics in medicine*, 21(22):3465–3474, 2002.

[19] Radford M. Neal and Jianguo Zhang. High dimensional classification with bayesian neural networks and dirichlet diffusion trees. In *Feature Extraction*, volume 207 of *Studies in Fuzziness and Soft Computing*, pages 265–296. Springer Berlin Heidelberg, 2006.

[20] Deepti Pachauri, Chris Hinrichs, Moo K. Chung, Sterling C. Johnson, and Vikas Singh. Topology-based kernels with application to inference problems in alzheimer's disease. *IEEE Transactions on Medical Imaging*, 31(10):1760–1770, 2011.

[21] Jim O. Ramsay and Bernard W. Silverman. *Functional data analysis*. Springer, New York, 2005.

[22] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Advanced Lectures on Machine Learning*, volume 3176, pages 63–71. Springer Berlin Heidelberg, 2004.

[23] Brian D. Ripley. Neural networks and related methods for classification. *Journal of the Royal Statistical Society. Series B (Methodological)*, 56(3):409–456, 1994.

[24] Mikail Rubinov and Olaf Sporns. Complex network measures of brain connectivity: Uses and interpretations. *NeuroImage*, 52(3):1059–1069, 2010.

[25] William A. Sethares and Ryan Budney. Topology of musical data. *Journal of Mathematics and Music*, 8(1):73–92, 2014.

[26] Robert H. Shumway and David S. Stoffer. Spectral analysis and filtering. In *Time Series Analysis and Its Applications*, Springer Texts in Statistics, pages 173–265. Springer New York, 2011.

[27] Cornelis J. Stam, Michael Breakspear, AnneMarie V. C. van Walsum, and Bob W. van Dijk. Nonlinear synchronization in EEG and wholehead MEG recordings of healthy subjects. *Human brain mapping*, 19(2):63–78, 2003.

[28] Ryota Tomioka, Kazuyuki Aihara, and Klaus-Robert Müller. Logistic regression for single trial EEG classification. In *Advances in Neural Information Processing Systems 19*, pages 1377–1384. MIT Press, 2007.

[29] Dongli Zhou, Wesley K. Thompson, and Greg Siegle. MATLAB toolbox for functional connectivity. *NeuroImage*, 47(4):1590–1607, 2009.

[30] Xiaojin Zhu. Persistent homology: an introduction and a new text representation for natural language processing. In *Proceedings of the Twenty-Third International Joint Conferene on Artificial Intelligence*, 2013.

[31] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 67(2):301–320, 2005.