EEG PATTERN CLASSIFICATION USING SUBSPACE METHODS AND SIMPLE CLASSIFIERS

Charles Anderson[1,3], Michael Kirby[2]
[1] Department of Computer Science
[2] Department of Mathematics
[3] Program in Molecular, Cellular, and Integrative Neuroscience
Colorado State University, Fort Collins, CO 80523

The manner in which EEG signals are represented for analysis and classification greatly affects the results obtained. To-date, most BCI research has relied on frequency-based representations, such as the energy within particular bands of frequency. Another common representation is based on autoregressive models, for which the coefficients have been shown to be very useful in classifying EEG as to which mental task a subject is performing[1]. A third class of representations are subspace methods that produce linear transformations designed to optimize various aspects of the transformed signals. In this poster, we describe a recent comparative study of several subspace methods for EEG classification.

One of the more common subspace methods is the Karhunen-Loève transform, also referred to as principal component analysis and singular value decomposition (SVD). The SVD transform maximizes the mean-square projection of the data on lower-dimensional subspaces, thus finding directions in the original data space along which the projected data has the highest variance. Less well-known transforms are maximum signal fraction analysis (SFA), that optimizes the amount of signal retained when signals are superposed, and canonical correlation analysis (CCA), that determines transformations of two data sets that produce the strongest correlation.

We compare each of these transformations on data that has been augmented by the method of delays, whereby values from consecutive samples are concatenated into one sample. As illustrated by our results, the size of the delay greatly impacts the efficacy of the data representation. We do not use the transformed signals for classification. Instead, we calculate the transformation matrix for overlapping half-second windows of data and use a subset of the columns of the transformation matrices as our representation.

Figure 1 shows classification accuracies of test data using the three subspace methods to represent the data and using either linear discriminant analysis (LDA) or a k-nearest-neighbor (kNN) classifier. As can be seen in the first column of Figure 1, the classification of the EEG data using the right singular vectors of the SVD depends significantly on the number of lags, or delays, used. No time lagging results in very poor classification rates for any number of modes; superior results are found empirically for lag two data. Five modes are required to obtain classification rates over 90%. Interestingly, lagging the data beyond two data points actually degrades classification performance. Fisher's LDA appears to consistently out-perform the kNN method. This suggests that only a subset of employed parameters are actually performing the discrimination, an hypothesis that warrants further study.

Classification results using CCA are shown in the second column of Figure 1. These results are similar to the SVD results, in that samples with zero lags do not contain enough information to discriminate the two tasks while adding one lag increases the classification accuracy significantly. The third column of Figure 1 shows the multi-mode discrimination capacity of SFA representation. Surprisingly, classification of the SFA modes with no lags works well. This is the only method examined that had this feature. It is also interesting to note that for zero to two lags there is no degradation in performance as more modes are included in the representation. Since it is the *noise* that is contained in the later modes, one may conclude that the correlated signal characterized by later modes is neither helpful nor problematic for the classification task. Performance of kNN does degrade with additional modes suggesting that this method

is more sensitive to noise.

References

[1] Charles W. Anderson and David A. Peterson. Recent advances in EEG signal analysis and classification. In R. Dybowski and V. Gant, editors, *Clinical Applications of Artificial Neural Network*, chapter 8, pages 175–191. Cambridge University Press, UK, 2001.

Figure 1: Percent of test samples correctly classified by LDA and kNN classifiers for data with zero to four lags represented as SVD, CCA, and SFA transforms. Each row of graphs is for a different number of lags, starting with zero in the top row to four in the bottom row. The first column of graphs is for the SVD representation, the second column is for the CCA representation, and the third column is for the SFA representation. The horizontal axis in each graph is the number of modes used to perform the classification.

THE BF++ FRAMEWORK (THE BIOFEEDBACK SOFTWARE DEVELOPMENT KIT)

Luigi Bianchi[1,4], Fabio Babiloni[2], Marco Arrivas, Patrizio Bollero, Maria Grazia Marciani [1,3]
[1] Dip. Neuroscienze, University of Rome "Tor Vergata", ITALY
[2] Dip. Fisiologia Umana e Farmacologia, University of Rome "La Sapienza", ITALY
[3] IRCCS, Fondazione "S. Lucia", Rome, ITALY
[4] Brainware, Rome, ITALY

A problem that commonly arises while developing cognitive bio-feedback (CBF) systems for disabled people is that it is very difficult to reuse them in a wide range of pathological situations. Very often these systems are designed using a "bottom-up" approach that is that starting from the particular problem the whole system is developed. This kind of approach fails because optimal use of every subject residual capability could require modifying the whole system in a way that is incompatible with practical needs. Moreover, these systems are specific to a limited set of platforms so re-engineering them could be a huge task: what happens if one wants to port a Windows 95/98 CBF application that runs on a notebook to a smaller Pocket PC running Windows CE?

The aim of BF++ is to give support to the creation of CBF systems by implementing a cross-platform C++ framework. As it is implemented using only ANSI C++ features it is possible to recompile it on virtually any platform.

Such framework must:

- **minimize programming effort,** by providing a skeleton of a CBF application and other facilities such as DSP and matrix computation routines;
- be **independent** on the nature and number of the **biological signals** used;
- allow the **integration with existing devices** such as screen readers, tactile mice, text to speech engines, etc…;
- allow the **diffusion of the resources** (data, algorithms, etc.);
- maximize source and binary code reuse;
- be efficient, to guarantee good performances in a wide range of situations;
- allow the realization of **low-cost** systems;
- be **independent** on the hosting platform (software and hardware).

By grouping all the aspects that are common to all the CBF systems implementations it has been possible to use a "top-down" design approach which offers a much greater flexibility and that allows the creation of a framework that can dramatically speed up the realization of several systems. Four main different operating modalities were provided: Setup, Training, Testing and Run.

A generic CBF system has been decomposed into 6 main functional blocks (plus the subject):

Acquisition, Kernel, Feedback Rule, Patient Feedback, Persistent Storage and Operator User Interface. Some of them are divided into sub-elements. For example the Kernel module is composed itself of more than 10 sub-objects. Assembling then it is possible to create a virtually infinite set of CBF systems that can be recompiled under any OS for which an ANSI C++ compiler is available. Special care was taken with respect to efficiency: in the PocketPC case this is obviously a key issue, while using Workstation it is possible to use many classifiers simultaneously in order to improve the overall system performance. This last feature was easily provided using well-known design patterns.

All the timing issues are also encapsulated into objects and no particular effort is required: special functions are automatically invoked whenever a trial starts (the OnTrailStart functions family), whenever a computation in performed (the OnTrialCompute family), whenever a trial ends (the OnTrailEnd family) and eventually whenever a classification occurs. By simply overriding them it is possible to provide a feedback to the user. An internal score is also provided to automatically evaluate the overall system performances: in the Testing modality, for example, the system ask to the patient to perform a task (e.g. one of many mental tasks) through a randomization engine and then it tries to recognize it. As it knows which task was required and which one was the classified one, it is able to update an internal score automatically. This and many other functionalities are already implemented in the framework as they do not depend on the nature of the biological signals used. Also file I/O is supported using XML as file format.

This approach was used to develop many different BCI systems for the Win32 platform. Some of them were already ported on a PocketPC device running Windows CE allowing the realization of wearable BCI systems. Also simulations using intracortical recordings were successfully done as well as using the Linux platform. Finally a simulation was also done on the SmartPhone 2002 platform: this suggests that it is also possible to realize PaceMaker like biofeedback and BCI systems using BF++.