Predicting Finger Movement Using an Ensemble Machine Learning Approach

Peter Handjinicolaou

Electrical and Computer Engineering Department

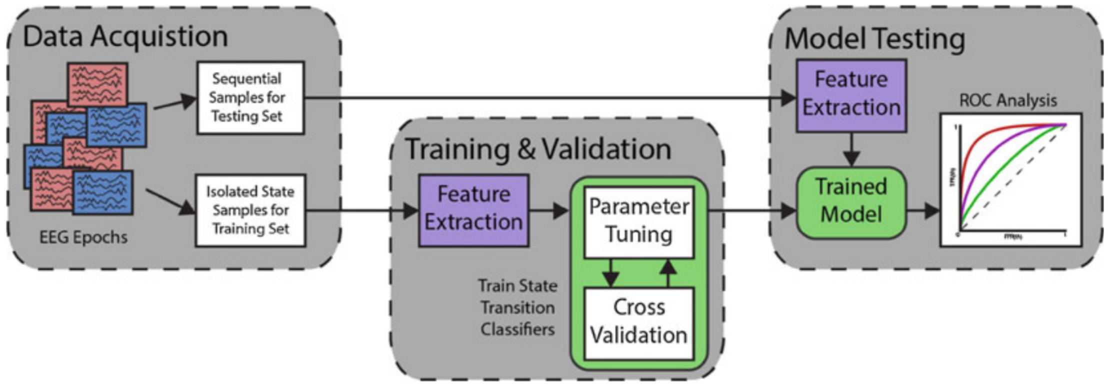
Abstract

Debilitating brain trauma caused by injury or stroke, and other neurological disorders, can hinder a person's ability to use their hands. Brain-Computer Interfaces (BCIs) are a subject of great interest regarding augmenting or restoring functionality to victims of this type of trauma. Motor imagery is a process in which a subject under test imagines performing an action without physically doing so [3]. Using brainwave sensors such as EEGs, the state of neural communication for that action can be recorded without the interference caused by the movement associated with it [8]. Using Machine learning classification techniques such as Support Vector Machines (SVMs), Multilayer Perception models (MLPs), and Fischer Linear Discrimination Analysis (LDA), it is possible to select for features and accurately predict upcoming movement by using motor imagery training and EEG data collection.

Introduction

For this paper we explore the recorded state of neural communication while a subject is typing with two fingers. The intended goal is to observe and eventually be able to predict an upcoming action, specifically, the pressing of a key by the left and right index fingers. Toward this end, an ensemble supervised classification approach developed by Mohammad Amin Alamalhoda and published to github as an open-source project [9]. It should be noted that this project is missing components, specifically Common Spatial Pattern (CSP) algorithms and was not functional out of the box. The missing components have been added by me.

Classification of EEG data requires a chain of processing that includes signal smoothing and filtering, feature extraction, feature discriminative analysis, and finally classification.



[4]

Figure 1) Block diagram of data acquisition, feature extraction, and classification

For this paper, data was obtained from the Berlin Brain-Computer Interfaces competition. consisting of 28 EEG channels aligned in accordance with the 10-20 system were sampled at 1000hz and low pass filtered under 200hz. For the purposes of reduction in processing time, the data was downsampled to 100hz, taking every 10th sample from the original dataset. The full dataset consists of both training and test data, and in both cases the subject is tapping 1 key per second per hand with their pinky and index fingers. For the training data, motor imagery techniques are used, and the subject does not physically move their index finger [3]. This data is labeled as left- and right-hand, and the associated labels are used in training the classifier. For this paper we aim to assess the use of CSP algorithms in classification accuracy, as well as the affect that different features have on successful classification. The criteria of success for this classification approach is a single output indicating left or right hand use.

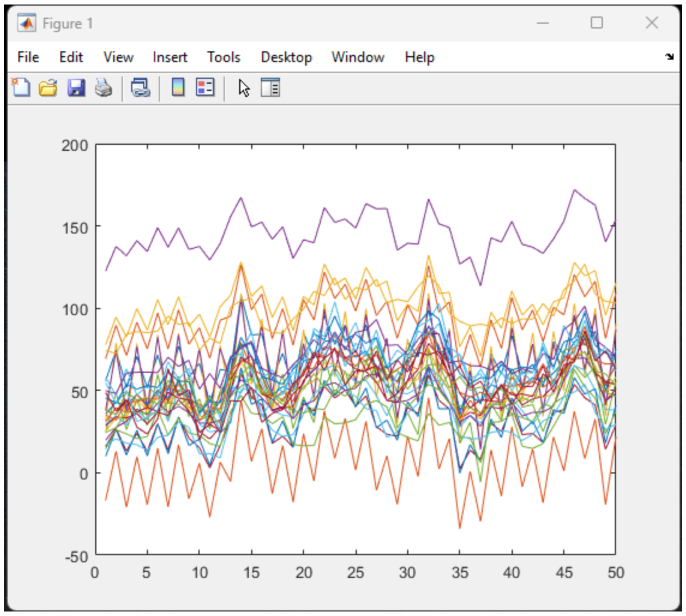


Figure 1) All 28 EEG sensors overlayed onto a single chart

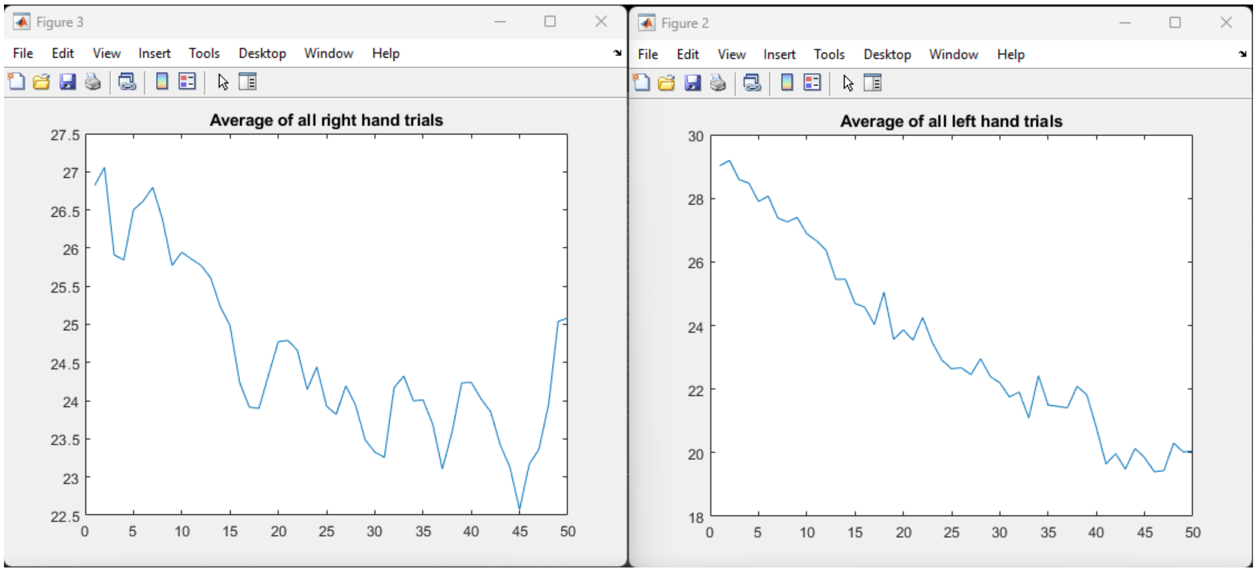


Figure 2): Average time series data from all 28 probes with left- and right-hand labeling.

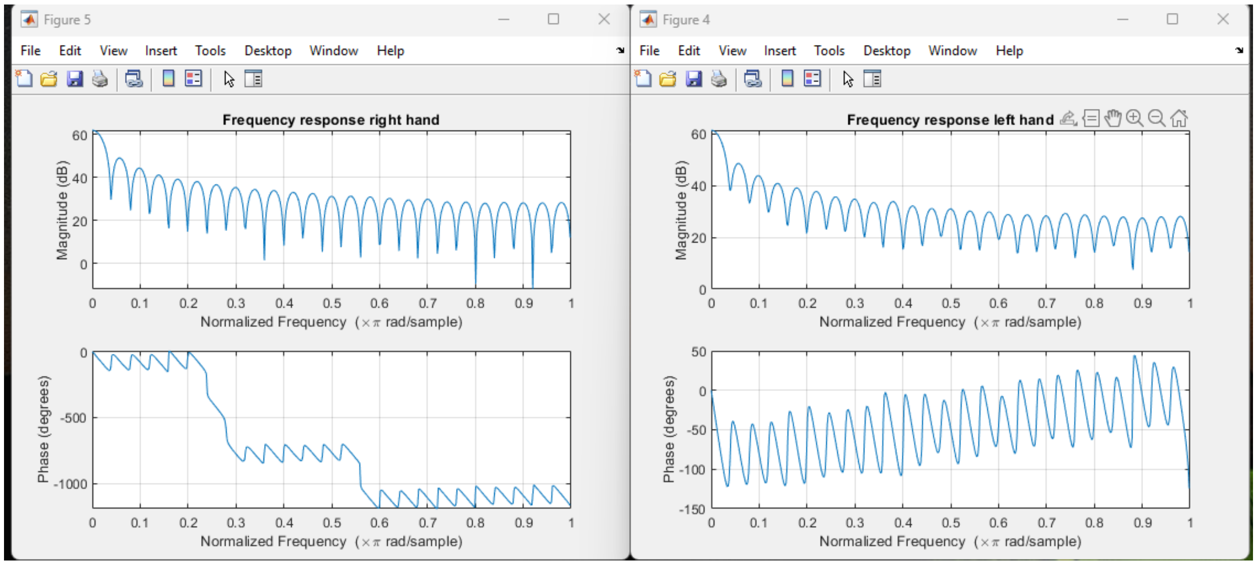


Figure 3) Frequency response data. Please note that once filtered, there is a 100x reduction in power associated in relevant frequencies compared to filtered frequencies (from ~60db to ~40db)

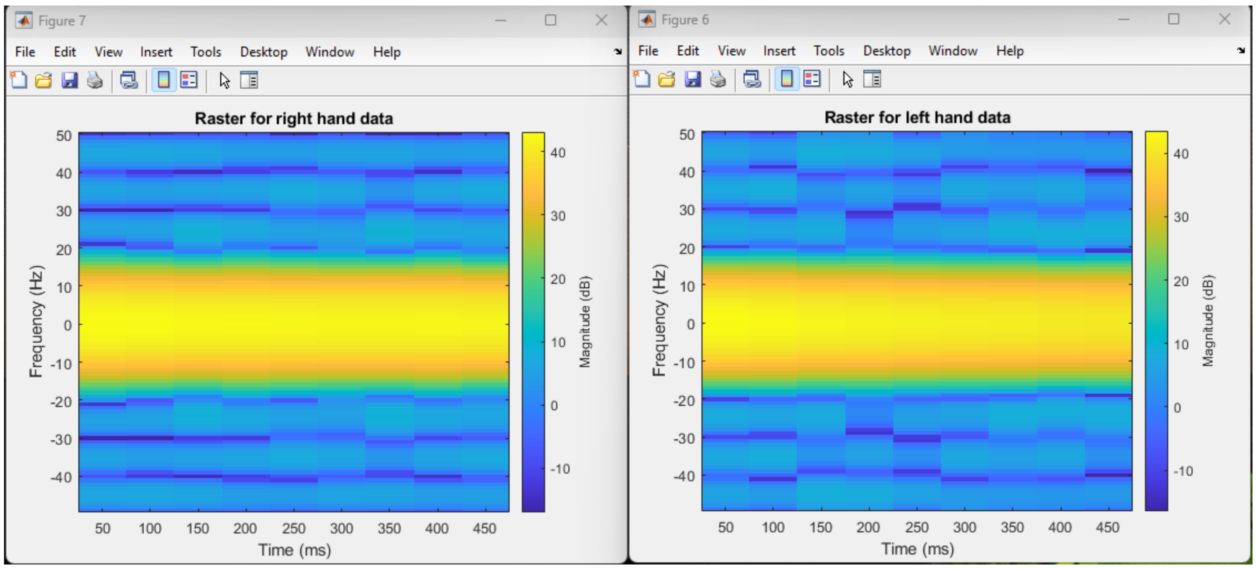


Figure 4) Raster/Waterfall view of frequency response for each hand. This view shows the variance of the frequency of the signal over time

Methods

Data preprocessing using CSP algorithms is common with multivariable signals. In a nutshell, CSP algorithms will dimensionally separate labeled data by account for maximum variance for one set of data and minimal variance for the second set. The goal of this approach is the optimal discrimination of two sets of data that otherwise may have significant overlap [10]. The algorithm for CSP used can be seen below:

```
%Common Spacial Pattern
X = x_train;
C_x1 = 0;
for i = left_idx
    C_x1 = C_x1+ X(:,:,i)'*X(:,:,i);
end

C_x2 = 1;
for i = right_idx
    C_x2 = C_x2+ X(:,:,i)'*X(:,:,i);
end
```

```
[V,D] = eig(C_x1, C_x2);
D = diag(D);
[D, indx] = sort(D, 'descend');
V = (V(:,indx));
W1 = V(:,1);
Wend = V(:,end);
X_filt = X;
```

The result of the addition of CSP to the classification algorithm appeared to improve results dramatically, but this will be discussed in the results section.

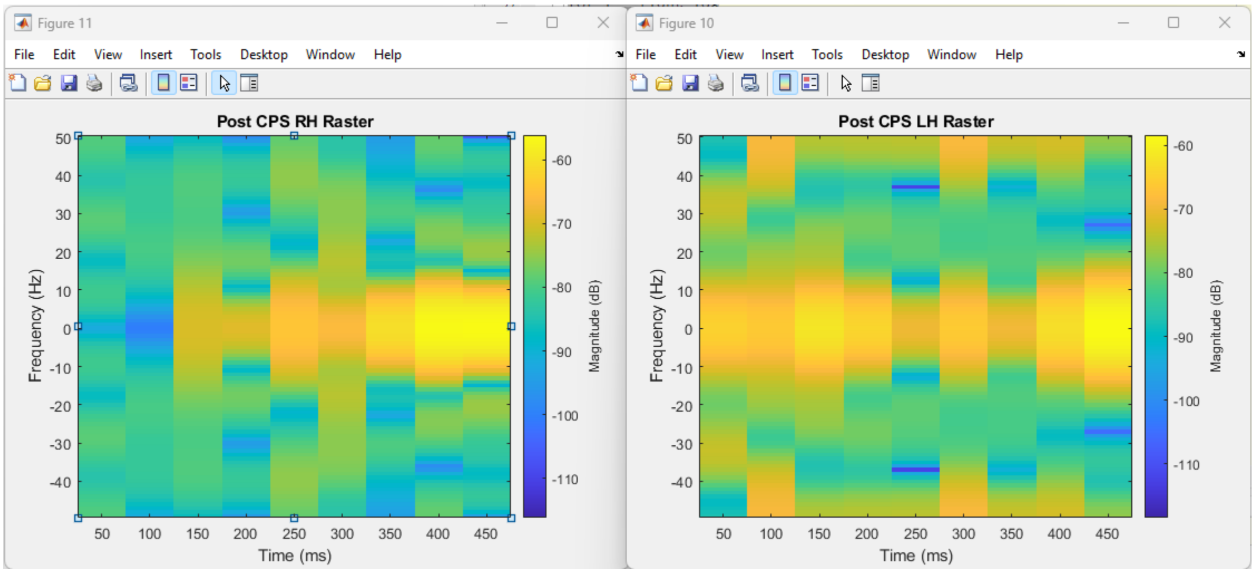


Figure 5) Raster view after Common Spatial Pattern algorithms are used. Please Note that compared to the raster plots without CSP, there is significantly less overlap and more distinguishable values.

Feature Extraction is the first step of any classification process. In machine learning, it is the process of building derived statistical values from an initial dataset. Feature extraction can be thought of as a transformation of a signal from raw data to a series of behaviors that accurately represent the data. For time and frequency domain analysis, the simplest EEG features are simple statistical derivations [5].

For this dataset, the following features were used: Mean and Median Frequency, Band power for Alpha, Beta, Delta, and Theta bands, variance, skewness, kurtosis, entropy, and lyapunov exponential. Please note that for the purposes of this paper, each feature was extracted using native MATLAB libraries.

1) Mean Frequency: The mean frequency of a spectrum can be quantified as follows: $f_{mean} = \sum_0^n I_i * f_i / \sum_0^n I_i$
2) Median Frequency: The median normalized frequency of the power-spectral-density of the signal $I = \sum_0^n I_i/2$ where $f_{median=} f_n$ *where* $[f_0 - f_n] > I$
3) Band power: The power spectral density across specific frequency bands: $P = 2\int_{f1}^{f2} S(f)df$ where S(f) is the spectrum of the signal.
4) Variance: The time domain variance of the sample values: $var = \frac{(n-1)X_{n-1} + X_n}{x}$
5) Skewness: Skewness can be defined as the measure of the asymmetry of a probability distribution relative to the mean value of a Random Variable and can be calculated as: $skewnes = E[\left(\frac{X-\mu}{\sigma}\right)^3$
6) Kurtosis: Another method of describing the "shape" of a probability distribution, focused on "tailed-ness" and can be quantified so: $kurtosis = E[\left(\frac{X-\mu}{\sigma}\right)^4$
7) Entropy: A statistical measure of randomness quantified with: $H(X) = -\sum_{i=1}^n P(x_i) \log P(x_i)$
8) The Lyapunov exponent can be represented with the following equation: $\lambda = \lim_{t\to\infty} \lim_{|\delta Z_0|\to 0} \frac{1}{t} \ln \frac{|\delta Z(t)|}{|\delta Z_0|}$

Feature Selection is the process by which the initial set of features is minimized and optimized to include only the most discriminatory features. The purpose of this process is to reduce dimensionality in the data set, which in turn reduces the complexity of calculation and compute time required. In order to select for the best features, and since we have two vectors of data, a 2-Dimensional Fisher Score was calculated in MATLAB using the following algorithm:

```
u0 = mean(all_data,'all');
u1 = mean(left_hand_data,'all');
u2 = mean(right_hand_data,'all');
var1 = var(left_hand_data);
var2 = var(right_hand_data);
Score = (abs(u0-u1)^2+abs(u0-u2)^2)/(var1+var2);
[9]
```

All features with a score less than twice the average were discarded. The remaining features were passed through an N-Dimensional Fischer Score calculator for final feature selection using the following algorithm implemented in MATLAB:

```
mu_0 = mean(all_sub')';
mu_1 = mean(sigg_cls1')';
mu_2 = mean(sigg_cls2')';
S1 = 0;
S2 = 0;

for sub1 = 1:size(sigg_cls1, 2)
    S1 = S1+(sigg_cls1(:,sub1)-mu_1)*(sigg_cls1(:,sub1)-mu_1)';
end
S1 = S1/sub1;

for sub2 = 1:size(sigg_cls2, 2)
    S2 = (sigg_cls2(:,sub2)-mu_2)*(sigg_cls2(:,sub2)-mu_2)';
end
S2 = S2/sub2;

Sw = S1+S2;
Sb = ((mu_1-mu_0)*(mu_1-mu_0)'+(mu_2-mu_0)*(mu_2-mu_0)');

Score = trace(Sb)/trace(Sw);
[9]
```

The classifier provided used two types of Artificial Neural Networks (ANNs) to train the model and use it to classify. A Multilayered Perceptron model (MLP) was used for the purposes of training, and a Radial Basis Function (RBF) was used for validation.

MLP networks are an expansion of a hardware prototype developed by Frank Rosenblatt known as the perception. The original perception could produce a single output from multiple inputs, for which a linear combination was formed. It can be described mathematically with the following formula: $\varphi(\sum_{i=1}^{n} w_i x_i + b) = \varphi(w^T x + b)$ [13].

As the name suggests, an MLP based ANN has multiple layers and are constructed by multiple perceptrons. An MLP is typically comprised of an input layer, one or more hidden layers, and an output layer. The hidden layers are where the computational gusto of this particular ANN can be found. These hidden layers are non-linear and as such seemingly small changes to the input layer can result in drastically different outcomes [11]. MLPs engage in supervised learning classification. When outputs are classified the model can check the accuracy of its prediction and map its error surface. Learning – or training – happens when the weights and biases of the model are modified based on the amount of error [12]. It should be noted that one flaw of MLP systems is that they can be over fitted to specific data [11]. Below is a basic example of an MLP diagram, and the provided algorithm for the MLP used in this project can be found in the code in appendix I.

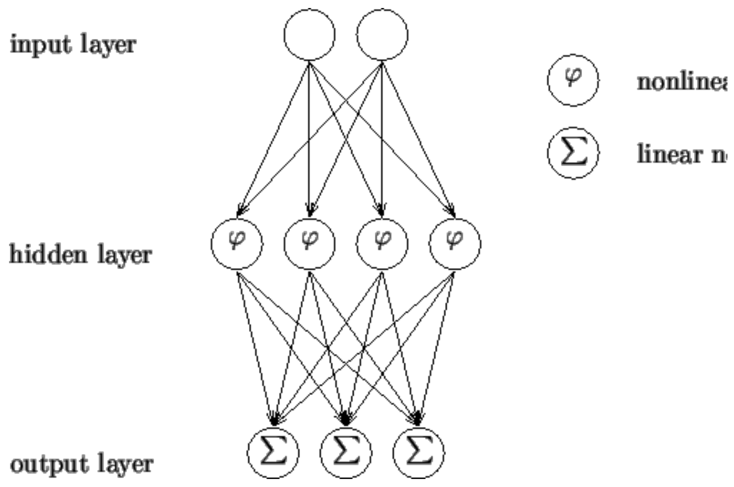


Figure 6) Multilayer Perceptron Block diagram. Please note the multiple neurons for input and the non-linear hidden layers. Figure provided by Annti Honkela [12]

As noted previously, the success criteria for our classification model resulted in single outputs per sample which would be either labeled left-hand, or right hand. The reader will note that an MLP network provides multiple outputs. For the final step of classification, the outputs of the MLP network are fed as inputs into an RBF network, which produces a single output.

First created by Broomhead and Lowe, researchers at the Royal Signals and Radar Establishment, RBF networks are functionally very similar to MLPs, relying on a linear combination of multiple inputs alongside hidden layers in order to create the output. A basic RBF can be described mathematically as follows: $\varphi(x) = \left(\sum_{i=1}^{N} a_i \rho(||x - c_i||)\right)$ [14]. The implemented algorithm is provided in appendix II.

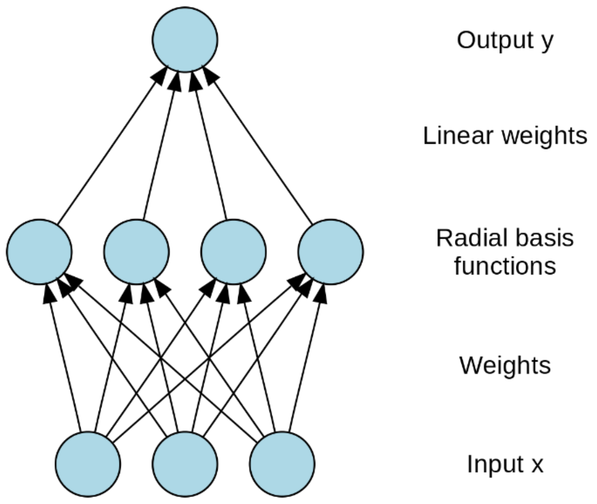


Figure 7) A simple RBF diagram [13]

Results & conclusions

Included in this paper are three sets of results. The first set of results does not include my implementation of CSP algorithms. The second set of results includes CSP, and the third set of results includes CSP with additional features added – specifically, entropy, kurtosis, and the Lyapunov exponent.

For the first set – characterized by figures 8 and 9 - the results are extremely poor. At only 52% accuracy in training and 49% accuracy in validation, it can be stated that this model essentially outputs random noise. Another indication of the lack of quality in this model is the divergence in the loss value between training and evaluation. In a higher quality model we would expect these to converge. Even with upwards of 50 generations of training, there is no discernable improvement.

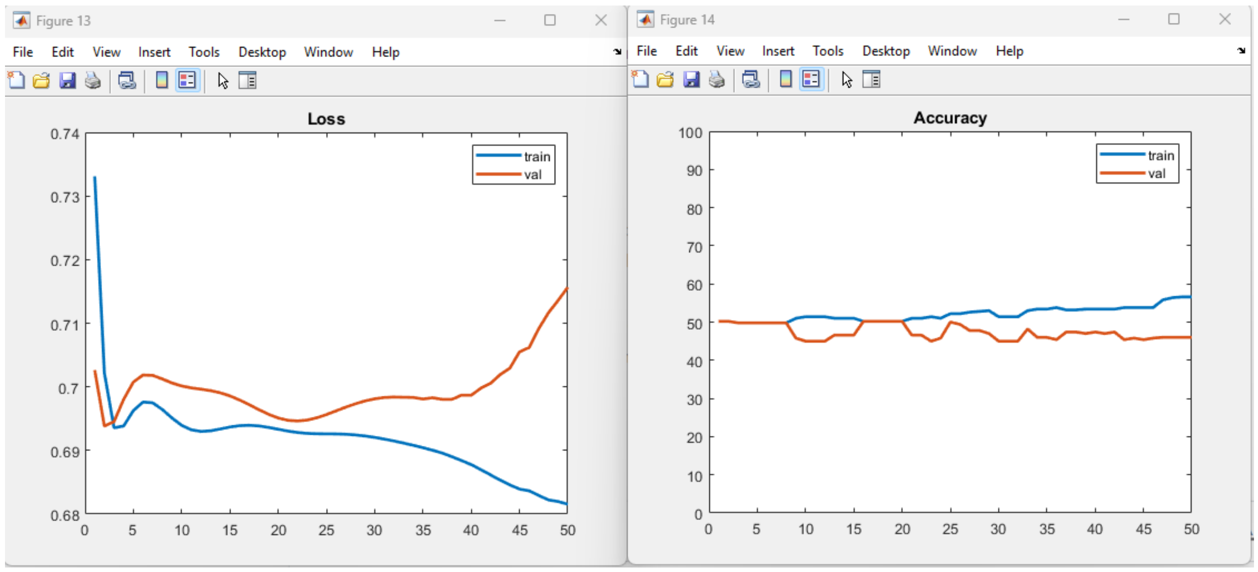


Figure 8) 10 generations of training without using CSP. Features used are: Mean frequency, median frequency, variance of the time series, and skewness. Left indicates loss per sample, Right indicates accuracy of prediction.

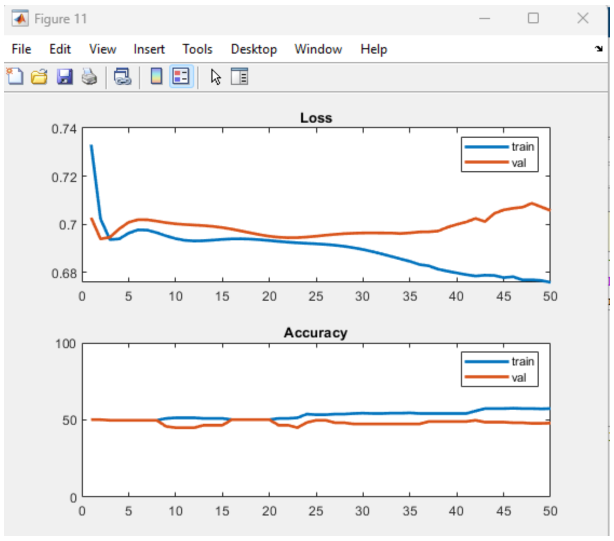


Figure 9) 50 generations of training without using CSP. Features used are the same as figure 8.

As expected, the introduction of CSP algorithms improves the results substantially. While 10 generations of training does not have a significant effect on total accuracy, it is clear that loss divergence is reduced. Accuracy could still be considered “random” at this point, however when we look at the results of 50 generations of training we begin to see positive results. Figure 11 shows us that loss divergence is minimized alongside loss being reduced, and accuracy rises to around 70%. It is possible that with more generations of training, higher accuracy could be achieved with the current set of features, however runtime on this model at 50 generations is roughly 8 hours of compute time – so a better solution needs to be found.

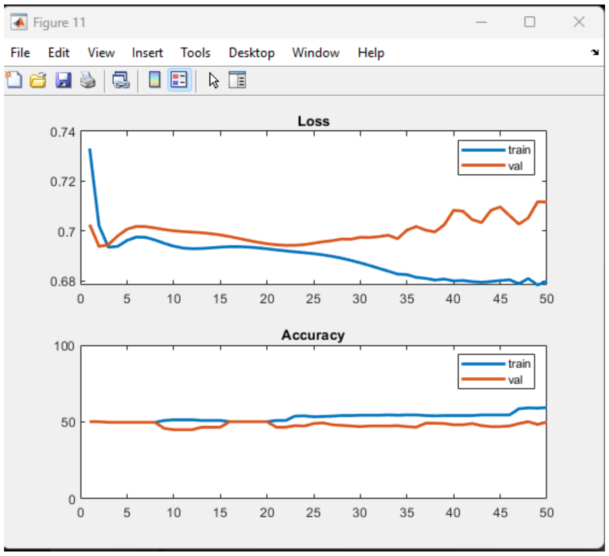


Figure 10) 10 Generations of training using CSP. Features used are the same as figure 8

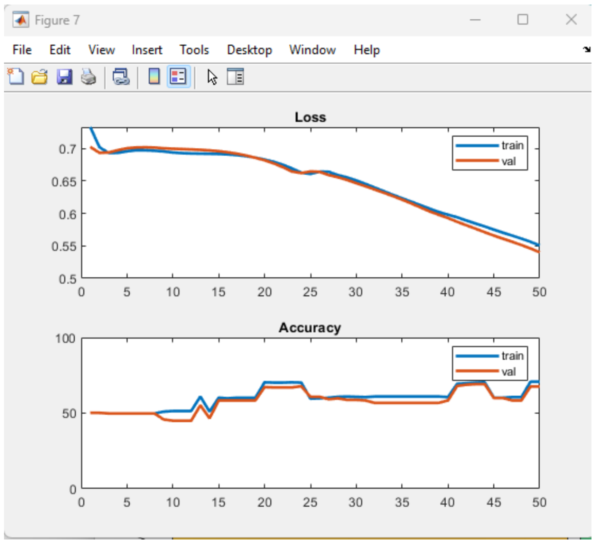


Figure 11) 50 Generations of training using CSP. Features used are the same as figure 8

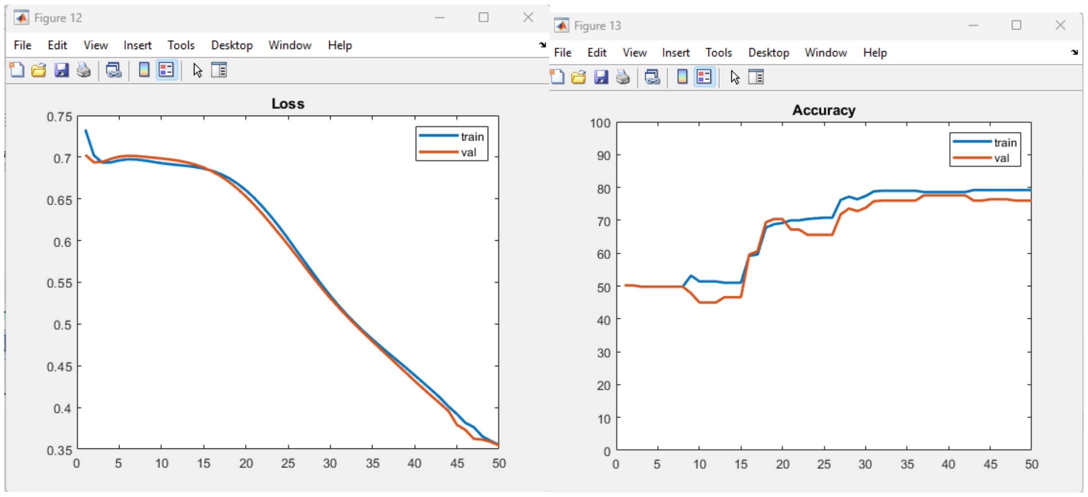


Figure 12) 10 Generations of training using CSP and additional features

Finally, our last set of results, shown above, includes added features previously mentioned. Due to computational time required, these features were only tested with 10 generations of training, but the results are significantly improved – showing accuracy levels of over 80%. From these results we can draw the following conclusions: 1) Predicting behavior based on EEG data using an ensemble classification approach is possible but processing intensive. 2) The use of common spatial patterning is beneficial in reducing error amongst datasets with large overlap. 3) Feature selection is extremely important toward classification accuracy.

With more time, it would be beneficial to see if there are more features that could be added or removed to reduce complexity in computation. This paper is also limited insomuch as no safeguards against overfitting the model were employed, nor were the results weighed with any kind of cross-correlation to detect if classification accuracy was a result of over fitted models.

References

1) B Blankertz, G Curio, KR Muller, “Classifying Single Trial EEG: Towards Brain Computer Interfacing” , 2001 , https://proceedings.neurips.cc/paper/2001/file/2d579dc29360d8bbfbb4aa541de5afa9-Paper.pdf
2) Berlin Brain-Computer Interface, Competition II, 2003, https://www.bbci.de/competition/ii/
3) B Blankertz, G Curio, KR Muller, “Data set <self paced 1s> , 2003 , https://www.bbci.de/competition/ii/berlin_desc.html
4) B He, “Neural Engineering 3rd Edition”, 2020, Department of Biomedical Engineering, Carnegie Mellon University
5) I Stancin, M Cifrek, A Jovic, “A Review of EEG Signal Features and Their Application in Driver Drowiness Detection Systems, 2021, National Library of Medicine ,https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8198610/#:~:text=The%20simplest%20features%20of%20the,%2C%20and%20similar%20%5B50%5D.
6) X Xiao, Y Fang, “Motor Imagery EEG Signal Recognition Using Deep Convolutional Neural Network”. 2021, Frontier Neuroscience, https://www.frontiersin.org/articles/10.3389/fnins.2021.655599/full
7) M Vilela, L Hochberg, “Applications of brain-computer interfaces to the control of robotic and prosthetic arms”, 2020, National Library of Medicine, https://pubmed.ncbi.nlm.nih.gov/32164870/
8) A Vahid, M Muckschel, S Stober, AK Stock, C Beste, “Applying deep learning to single-trial EEG data provides evidence for complementary theories on action control”, 2020, Nature News, https://www.nature.com/articles/s42003-020-0846-z
9) M A Alamalhoda, “EEG-Classification”, https://github.com/MohammadAminAlamalhoda/EEG-Classification
10) Z Koles, M Lazar, S Zho, “Spatial patterns underlying population differences in the background EEG”, 1990, Nature News, https://link.springer.com/article/10.1007/BF01129656 - CSP paper
11) R Keim, “How to Train a Multilayer Perception Neural Network”, 2019, Allaboutcircuits.com, https://www.allaboutcircuits.com/technical-articles/how-to-train-a-multilayer-perceptron-neural-network/
12) A Honkela, “Multilayer Perceptrons”, 2001 , http://users.ics.aalto.fi/ahonkela/dippa/node41.html
13) C McCormick, “Radial Basis Function Network Tutorial”, 2013, https://mccormickml.com/2013/08/15/radial-basis-function-network-rbfn-tutorial/
14) D S Broomhead, D Lowe, “Multivariable Functional Interpolation and Adaptive Networks”, 2001, Royal Signals and Radar Establishment, https://sci2s.ugr.es/keel/pdf/algorithm/articulo/1988-Broomhead-CS.pdf

Appendix I:

```
function [mean_loss_train, mean_loss_val, mean_acc_train, mean_acc_val] = MLP(datas,
K_folds, lr, num_epochs, y_train, k, fs, num_itters)

    num_datas_in_fold = floor(size(datas, 3)/K_folds);
    num_all_datas = size(datas, 3);
    loss_train = [];
    loss_val = [];
    acc_train = [];
    acc_val = [];

    for f = 0:K_folds-1

        if f == 0
            folds_train_1 = [];
            folds_val = f*num_datas_in_fold+1:(f+1)*num_datas_in_fold;
            folds_train_2 = (f+1)*num_datas_in_fold+1:num_all_datas;
        elseif f == K_folds-1
            folds_train_1 = 1:f*num_datas_in_fold;
            folds_val = f*num_datas_in_fold+1:(f+1)*num_datas_in_fold;
            folds_train_2 = [];
        else
            folds_train_1 = 1:f*num_datas_in_fold;
            folds_val = f*num_datas_in_fold+1:(f+1)*num_datas_in_fold;
            folds_train_2 = (f+1)*num_datas_in_fold+1:num_all_datas;
        end
        folds_train = [folds_train_1, folds_train_2];

        datas_train = datas(:, :, folds_train);
        datas_val = datas(:, :, folds_val);

        [~,cls1_index] = find(y_train(:,folds_train) ==0);
        [~,cls2_index] = find(y_train(:,folds_train) ==1);
        %Common Spacial Pattern
        X = datas_train;
        C_x1 = 0;
        for i = cls1_index
            C_x1 = C_x1+ X(:,:,i)'*X(:,:,i);
        end

        C_x2 = 0;
        for i = cls2_index
            C_x2 = C_x2+ X(:,:,i)'*X(:,:,i);
        end


        [V,D] = eig(C_x1, C_x2);
        D = diag(D);
        [D, indx] = sort(D, 'descend');
        V = (V(:,indx));
        W1 = V(:,1);
        Wend = V(:,end);

        %%experimental - look into W1 / Wend


        %W_Filt = CSP(datas_train, y_train(:,folds_train));

        datas_train_filt = zeros(size(datas_train, 1), size(V, 2), size(datas_train,
3));
```

```
        datas_val_filt = zeros(size(datas_val, 1), size(V, 2), size(datas_val, 3));

        for i = 1:size(datas_train, 3)
            datas_train_filt(:,:,i) = (V'*datas_train(:,:,i)')';
        end

        for i = 1:size(datas_val, 3)
            datas_val_filt(:,:,i) = (V'*datas_val(:,:,i)')';
        end

        [features_train, num_features] = feature_extraction(datas_train_filt, fs);
        [features_val, num_features] = feature_extraction(datas_val_filt, fs);

        [~, cls1_indexes] = find(y_train(folds_train)==0);
        [~, cls2_indexes] = find(y_train(folds_train)==1);

        [selected_features, row_selected, col_selected] =
fisher_2D_calculator(num_features, features_train, cls1_indexes, cls2_indexes);


        [best_features_train, best_features_indexes] = fisher_ND_calculator(k,
num_itters, selected_features, cls1_indexes, cls2_indexes);

        features_index_col = col_selected(best_features_indexes);
        features_index_row = row_selected(best_features_indexes);
        best_features_val = zeros(length(features_index_row), size(datas_val, 3));
        for i = 1:length(features_index_row)
            best_features_val(i, :) = features_val(features_index_row(i), :,
features_index_col(i));
        end


        datas_train = py.numpy.array(best_features_train');
        datas_val = py.numpy.array(best_features_val');
        labels_train = py.numpy.array(y_train(:,folds_train)');
        labels_val = py.numpy.array(y_train(:,folds_val)');

        kwa = pyargs('datas_train', datas_train, ...
            'datas_val', datas_val, ...
            'labels_train', labels_train, ...
            'labels_val', labels_val, ...
            'input_size', int32(k), ...
            'ouput_size', int32(1), ...
            'lr', double(lr), ...
            'num_epochs', int32(num_epochs));

        mod = py.importlib.import_module('classifier');
        py.importlib.reload(mod);
        out = mod.call_from_matlab(kwa);

        loss_train = [loss_train; str2num(char(out.cell{1}))];
        loss_val = [loss_val; str2num(char(out.cell{2}))];
        acc_train = [acc_train; str2num(char(out.cell{3}))];
        acc_val = [acc_val; str2num(char(out.cell{4}))];

    end

    mean_loss_train = mean(loss_train, 1);
    mean_loss_val = mean(loss_val, 1);
    mean_acc_train = mean(acc_train,1);
    mean_acc_val = mean(acc_val,1);
```

Appendix II

```
function [acc_train, acc_val] = RBF(K_folds, best_features, y_train)


    acc_train = [];
    acc_val = [];

    k = size(best_features, 1);
    num_datas_in_fold = floor(size(best_features,2)/K_folds);
    num_all_datas = size(best_features,2);


    for f = 0:K_folds-1

        if f == 0
            folds_train_1 = [];
            folds_val = f*num_datas_in_fold+1:(f+1)*num_datas_in_fold;
            folds_train_2 = (f+1)*num_datas_in_fold+1:num_all_datas;
        elseif f == K_folds-1
            folds_train_1 = 1:f*num_datas_in_fold;
            folds_val = f*num_datas_in_fold+1:(f+1)*num_datas_in_fold;
            folds_train_2 = [];
        else
            folds_train_1 = 1:f*num_datas_in_fold;
            folds_val = f*num_datas_in_fold+1:(f+1)*num_datas_in_fold;
            folds_train_2 = (f+1)*num_datas_in_fold+1:num_all_datas;
        end
        folds_train = [folds_train_1, folds_train_2];


        datas_train = best_features(:,folds_train);
        datas_val = best_features(:,folds_val);
        labels_train = y_train(:,folds_train);
        labels_val = y_train(:,folds_val);

        net = newrbe(datas_train,labels_train, 'spread', 4);
        val_out = net(datas_val);
        train_out = net(datas_train);
        [row, col] = find(floor(train_out)==labels_train);
        acc_train = [acc_train, length(row)/length(train_out)];
        [row, col] = find(floor(val_out)==labels_val);
        acc_val = [acc_val, length(row)/length(val_out)];


    end
```