Brain-Computer Interface for ALS Patients

Technion - Israel Institute of Technology
Electrical & Computer Engineering

CRML - Control Robotics and Machine Learning Laboratory

Roee Latzres Tomer Krichli

Instructor: Ben Kretzu

September 2023

Contents

Acronyms

1. ALS - Amyotrophic lateral sclerosis.
2. BCI - Brain-Computer interface.
3. EEG - Electroencephalogram.
4. ML - Machine Learning.
5. AI - Artificial Intelligence.
6. GA - Genetic Algorithm.
7. ICA - Independent Component Analysis.
8. DL - Deep Learning.
9. SVM - Support Vector Machine.
10. LDA - Linear Discriminant Analysis.
11. KNN - K-Nearest Neighbors.
12. NB - Naive-Bayes.
13. LR - Logistic Regression.
14. MV - Majority vote.
15. EA - Expert Advice.
16. HPF - High Pass Filter.
17. LPF - Low Pass Filter.
18. NCA - Neighborhood Component Analysis.

1 Abstract

ALS - Amyotrophic lateral sclerosis, also known as Lou Gehrig's Disease, is a rare neurological disease that affects motor neurons — nerve cells in the brain and spinal cord that control voluntary muscle movement. Voluntary muscles are those we choose to move to produce movements like chewing, walking, and talking.

As the disease progresses, weakness and atrophy spread to other parts of your body. However, the disease doesn't affect the different brain functions or the ability to taste, touch, smell or hear. As a result a patient might find himself unable to communicate with his or hers surroundings even though if the ability to receive signals from the environment is perfectly fine.

Due to the fact that the patients mind's remains clear thorough out the course of the disease, different systems of Brain-Computer Interfaces, BCIs, were developed to help patients communicate with the environment. However these systems might be expensive and unreliable.

In this project we attempt to use machine learning tools in order to develop a way for ALS patients to communicate with their surroundings, using an affordable BCI system, even if they lose control over their muscles.

To that purpose we've used an EEG based BCI system which makes use of electromagnetic waves emitted by the brain to record the brain activity and later on ML algorithms to predict the patients thoughts solely by using the recording.

2 Introduction

2.1 Background

As we've described before, at a certain point during the course of the ALS disease, the patients cannot communicate through the conventional ways healthy people communicate. The most common way, is of course, talking, but we can also include hand gestures, facial expressions, etc. The systems that patients use today are mainly based on eye movement, and there are some systems that use EEG based Brain Computer Interface (BCI) to help the patients to communicate with their surroundings. The main issue with such systems is their price, and sometimes (like in eye movement based systems) their lack of mobility. One of the main problems of EEG signals, is its non stationary nature. In most ML algorithms, we feel confident about data-sets being stationary, at least for a large set of features of that data. For example, we can assume stationary behavior for a pretty long term of time if we are looking at cars from the last 20 years. Yes, cars' look is changing through the years, but cars' main features are indeed stationary, like wheels, windows, side mirrors, handles, etc. When dealing with brain waves, and with EEG in particular, we cannot take such assumptions as realistic ones, since our data distribution can change every few weeks or even days.

2.2 Project Goal

Here we jump in to the rescue! We use a low cost home made BCI set that can record the EEG signals from the patient brain. Patient can make three different decisions, e.g. right, left, and center. Since in most of the cases of ALS the patient's cognitive abilities remain the same, using data from the brain might be very useful, and also help the patient express him/her-self better.

Bottom line, our goal is to use a low cost BCI set, that can handle the non-stationary behavior of EEG signals, using ML and AI algorithms, to classify between three choices of the patient. In our project we'll use a stimulation method that is called Motor Imagery (MI). In this method, the patient is required to think, i.e. imagine, a limb (hands / feet) or tongue movement, and the system will determine from the limb that moved, what message the patient wants to tell his surrounding.

2.3 Mode of Work

Our system is constructed from an affordable BCI set, that contains 11 electrodes. We use that set to record our data. After the data is recorded, we segment the data, preprocess it, and than extract features to be the input to our classifier. Then an AI algorithm is choosing the best features for our classifier, and a decision between left, right or idle is the output of our classifier. To sum it up - out system input is EEG signals' features, and the output is a decision out of the set {left, right, idle}.

3 Literature Review

Trying to make a classifier that is robust to the non stationary nature of EEG signal is a complex task. We need to know how to first pre-process the data, then what features to extract from the recordings. After all of that is done, we still have a lot of freedom degrees, where we need to choose our features to be the input to our classifier, and of course, choose the type of our classifier. After having a literature review, these are the common methods/approaches for each topic:

- Pre Process: The common, most basic approach is to use first LPF with $\omega_c = 0.5[Hz]$, then apply HPF with $\omega_c = 40[Hz]$ and finally use a notch filter using $\omega_c = 50[Hz]$ to remove artifacts caused by electric infrastructure (Note: the notch filter cutoff frequency depends on the electric infrastructure in your country). After removing the irrelevant frequencies, we've used a Laplacian filter over $C03$ and $C04$ electrodes, to decrease the spatial noise added to those electrodes by their neighbor electrodes. To make sure we are using the cleanest data, we apply Independent Component Analysis (ICA), presented in [3], to clean components like eye movement artifacts. Important note: the above pre-process method was applied on the whole data (all of the trials of one recording) at once.

- Feature Extraction: We first began with statistical basic features, such as Skewness, Kurtosis, and Entropy. We also extracted the band power of 5 different frequency bands: $[8, 10.5], [10, 15.5], [12.5, 30], [15.5, 18.5], [17.5, 20.5]$. To use the spatial properties of our electrodes, we used Common Spatial Patterns (CSP) features. We took the 3 most variant features to be used as features. Another features used were extracted by calculating the Pwelch matrix, extracting Spectral Moment, Root total power, spectral edge and entropy, slope, etc... All of the features we've extracted can be found in our code, and further explanation can be found in [3].

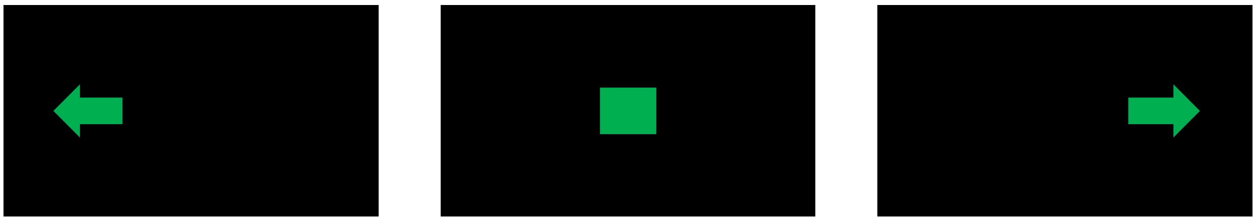
- Feature Selection: After creating a large feature space, a method was needed to narrow the features down to the most promising ones that will help us predict the correct output label that the patient thought about. We tried two different methods: Statistical analysis of past features and an AI algorithm named Genetic Algorithm (GA). After disappointing results from the Statistical analysis we focused our efforts in the GA approach. To put in a nutshell, the genetic algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation. In our case the individuals are the different features and in each step of the algorithm we try to choose the most promising combination of features to get the prediction right. Further explanation can be found in [4]

- Classifier selection: Since we do not have much data in our hands, we knew that the optimal solutions on the shelf of the ML store, i.e. Deep Learning tools such neural networks (NN) won't help us. Reading [3] helped us to filter and use the best "simple" classifiers, such as Support Vector Machine (SVM), Linear Discriminant Analysis (LDA), Naive-Bayes (NB) and K-Nearest Neighbors (KNN). We've decided to add few extra classifier, e.g. Logistic Regression (LR).
- Classifier Ensemble: Once we had the prediction of all of our classifiers, we realised we could use this data to further improve our prediction. In order to do that we tried two different methods - The first was Expert Advice (EA) [1]. EA uses more sophisticated approach. By assigning weights to each of our classifiers (or "experts") we are able to use the training data to adjust the weights and improve our prediction. We also added a 'Stacking' [5] classifier, which stacks 2 classifiers upon each other. The first classifier is an ensemble classifier, learning from the data, and the second is a classifier that learns from the results of the first level classifier.

4 Project Description

4.1 Data Acquisition

The main goal of this project is to create a system that is robust to the non-stationarity of the EEG signals from the brain. To that end, data that is spread of long period of times is required. Unfortunately we could not find online data that suited this criteria, hence we had to make our own data. For the span of 6 weeks we've recorded multiple sessions, 3 times a week from two different subjects (ourselves).
Each session is comprised of 60 trials in which the subject will be shown randomly one of 3 option - right, left or idle, as seen in Fig 2. A few seconds after the option is shown the subject must stimulate the corresponding side. Our goal is to be able to predict the side the subject has thought of using only the EEG signals.



(a) left arrow　(b) idle　(c) right arrow

Figure 2: The different option for a subject to choose from in each trial. After an image is shown to the subject, the subject will need to stimulate the corresponding area for a specified amount of time.

4.2 Block Scheme

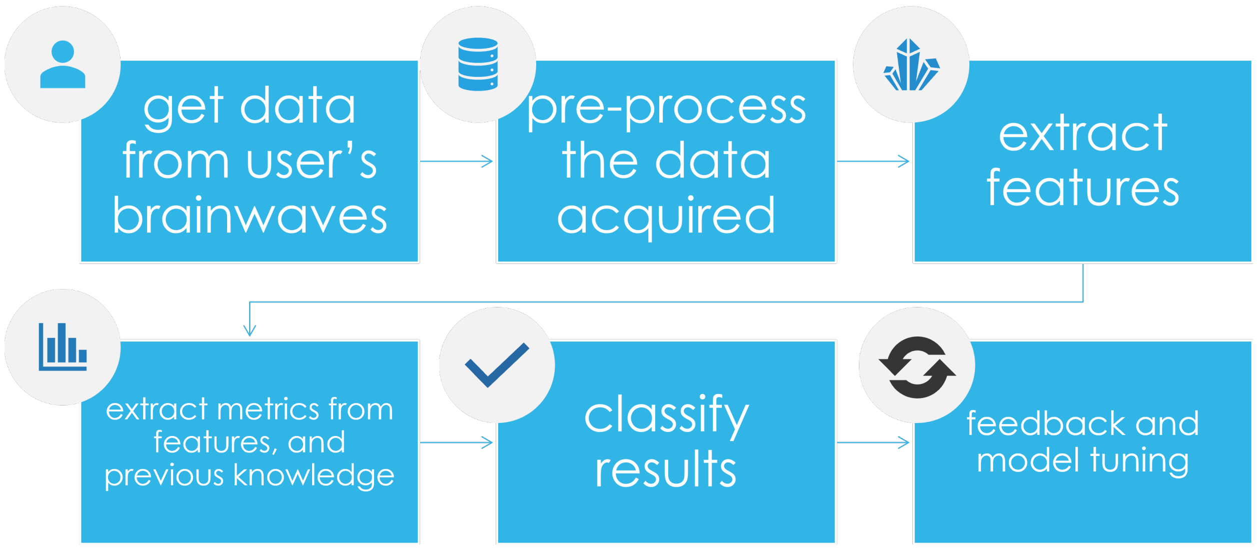

get data from user's brainwaves
pre-process the data acquired
extract features
extract metrics from features, and previous knowledge
classify results
feedback and model tuning

Figure 3: Top Level block diagram of our project, describing each block part in the pipeline.

As we can observe in Fig 3, our project contains 6 main blocks. We'll now describe each one of them, starting from the pre process stage, since we've already explained the data acquisition stage.

4.3 The Pipe Stages

4.3.1 Pre Process & Segmentation

The pre-process is being applied on the raw EEG time series, for each electrode (except #4 on the list). the stages of the pre process contains:

1. High pass filter (HPF) with $\omega_c = 40[Hz]$
2. Low pass filter (LPF) with $\omega_c = 0.5[Hz]$
3. Notch filter with $\omega_c = 50[Hz]$, to neutralize artifacts caused by the electrical infrastructure.
4. Laplacian filter applied on electrodes C03 & C04 only.
5. ICA applied, then components that classified as artifacts were removed from our data.

After the data is being pre processed, we are moving to the segmentation stage. The segmentation stage divides the time series to segments, and each segment it being labeled as on of the 3 classes we are trying to classify, (i.e. right, left, idle).

Note: segment is defined as interval of time, fixed in its length, that contains a sub-time-series including the data relevant for a trial classified as one of the 3 classes.

4.3.2 Features Extraction

After pre-processing the data, and segmenting it, we have a data-set ready to be feature extracted. Most of the features that were extracted were described in the literature review section, and all of the features extracted can be found in our code.

After extracting the features from the recording data, we got a matrix $X \in \mathbb{R}^{n\times m}$ where n is the number of trials (samples, segments), and m is the number of features.

4.3.3 Features Selection

In this section, we did most of our work in the first part of the project. We wanted to examine two methods for feature selection. One was our own invention semi-automatic method, and the other was fully algorithmic powered.

4.3.3.1 First Approach - Statistical Analysis

Assuming that with smaller time difference between each recording we will reduce non-stationarity in the data, we tried to extract features that will yield the best results using a different number of adjacent recordings. After pre-processing we have projected the values of each feature to the corresponding class (right & left) over all the trials we have conducted, and then applying linear regression, as can be visualized in Fig 4. From the regression results, we calculated different metrics such as:

- R^2 Score: $R^2(y,\hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i-\hat{y}_i)^2}{\sum_{i=1}^{n}(y_i-\bar{y})^2}$, where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$.
- $\text{MAE}(y,\hat{y}) = \frac{1}{n_{\text{samples}}}\sum_{i=0}^{n_{\text{samples}}-1}\left|y_i - \hat{y}_i\right|.$
- Pinball Loss: $\text{pinball}(y,\hat{y}) = \frac{1}{n_{\text{samples}}}\sum_{i=0}^{n_{\text{samples}}-1}\alpha\max(y_i-\hat{y}_i,0)+(1-\alpha)\max(\hat{y}_i-y_i,0)$

And the full list can be found in our code. All of the metrics implementations can be found in 'sklearn.metrics' library in python.

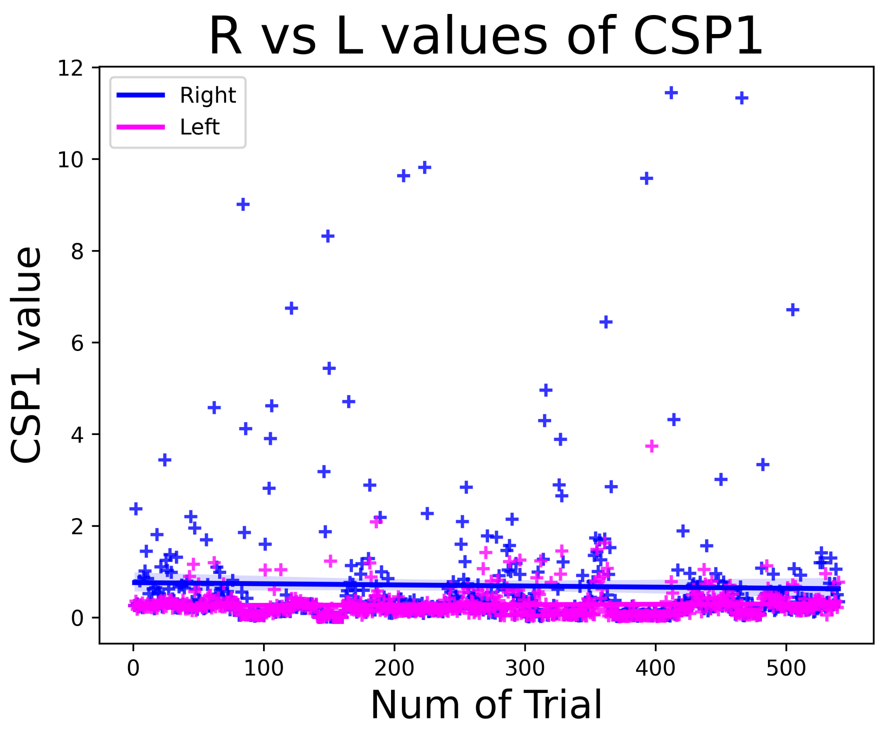


Figure 4: CSP1 feature value vs. all of the trials conducted in our product, with linear regression applied.

To determine the best features, we first began with testing different time constants, to determine which time constant will yield the best features in terms of classification results (given a basic R^2 *Score* metric to rank features). With that time constant chosen we ran through the different metrics to see which yields the best results.

4.3.3.2 Second Approach - Genetic Algorithm

Before jumping to the feature selection process, let's discuss a bit about the algorithm. The Genetic Algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation. In our case the individuals are features and we seek the best combination of individuals that will produce the best results given a predetermined classifier. Other special characteristic of the algorithm is mutation and crossover which can further improve the diversity of individuals chosen.
We start with a list of all our features and create several sets of randomly selected features from that list. Each set will comprise of 0s and 1s to represent whether a feature has been selected or not, for example if we had 10 features:

$$\underbrace{0011011001}_{set\ 1},\ \underbrace{0111000001}_{set\ 2},\ \ldots$$

We then evaluate our sets according to a fitness function, in our case the classification rate, to see which sets have the best performance. With the best sets that we currently have we can build the next generation of sets using two processes called crossover and mutation. Once we have created the next generation of sets we can repeat the process until we reach a predetermined threshold (certain amount of generations or certain score of the fitness function).

4.3.3.3 Chosen Approach

As can be seen in Fig. 5, the first approach (STA) is inferior for any classifier. Hence, we decided to drop it, and move forward only with GA feature selection method. It seems, that the GA features selection method is much more robust, and helps us to reduce our feature space.

4.3.3.4 Moving to Online Classifier

After choosing to move on with GA, we need to find a way to choose the best features to build an online model, i.e. a model that has fixed weights, and can work only with specific features. To do so, we've ran $\sim$ 70 times the GA process, and we've saved for each classifier which features were selected the most. With this approach, we've tried to have a look at the classifier GA selected features histogram, to start from a better start point when running the next GA, using a sort of feedback to reduce our feature space, and make the search problem smaller.
Using all of the above, we finally took all of the best features, using our feedback GA features selection method, and tailored for each classifier its best features.

4.3.4 Classifier Selection

After creating a sufficiently large enough feature space we needed a robust classifier that can handle the non-stationary data.

We started out by evaluating the results of the basic classifiers using the features selected in the feature selection phase (such as SVM, LDA, NB, KNN etc) as can be seen in 5.

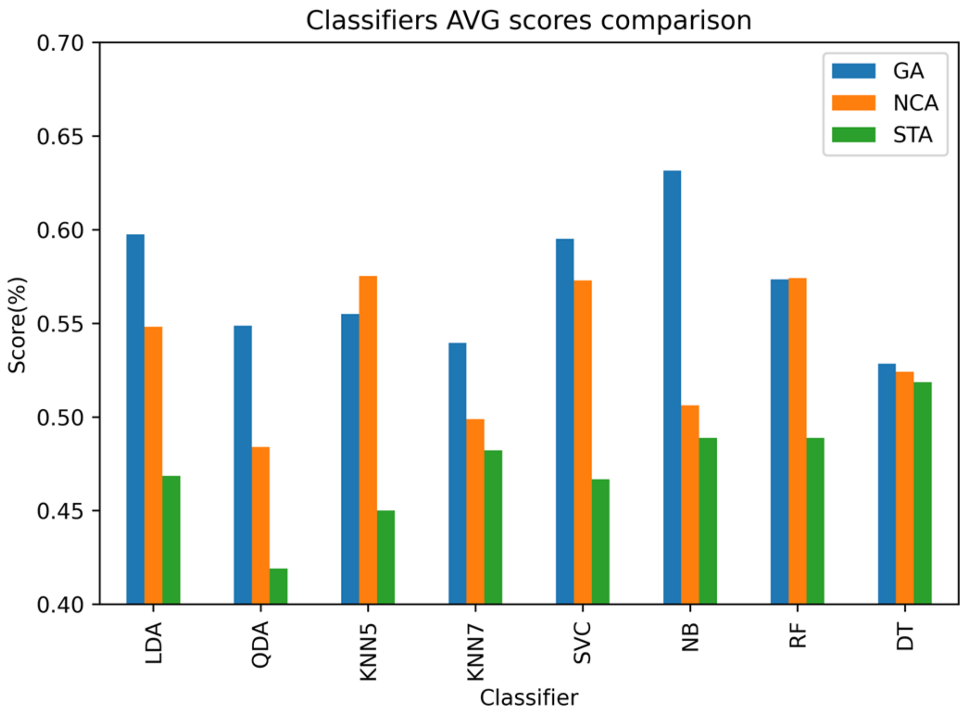


Figure 5: Average score on different basic classifiers

Once we had all the basic classifiers results we looked into how we can improve our prediction success rate by employing various techniques that would make use of the predictions we already had from the basic classifiers such as: Majority vote, Expert Advice and Stacking.

- Majority Vote: each basic classifier provides a vote as for which prediction it believes is the correct prediction and the prediction is decided by the majority.
- Expert Advice: referring to each classifier as an expert and assigning weights to each of them using the prediction of the training data.
- Stacking: in this approach we use a two level system for our models. The first level is called Level-0 Models, the models in this level fit on the training data regularly. The second level is called Level-1 Models, here we create a Model that learns how to best combine the prediction of all the Level-0 models.

Important note: in our project we dropped the "Vanilla" Majority Vote, and we refer Majority Vote as Expert Advice.

4.4 Optimization and Data Cleansing

The last part of our project was to try and squeeze out whatever we could from our system in order to get the best predictions. To that end we tried to, firstly, optimize the GA we used on our data and, secondly, removing noisy data that was harming our system i.e. bad recordings.

- Optimization: In order to optimize our GA we used a dedicated Python framework named "Optuna". This framework help automate the search for optimal hyper-parameters to be used on the desired algorithm by running many consecutive trails with a specific search space. In addition the framework provides insights to the importance of the parameters that are being tracked.

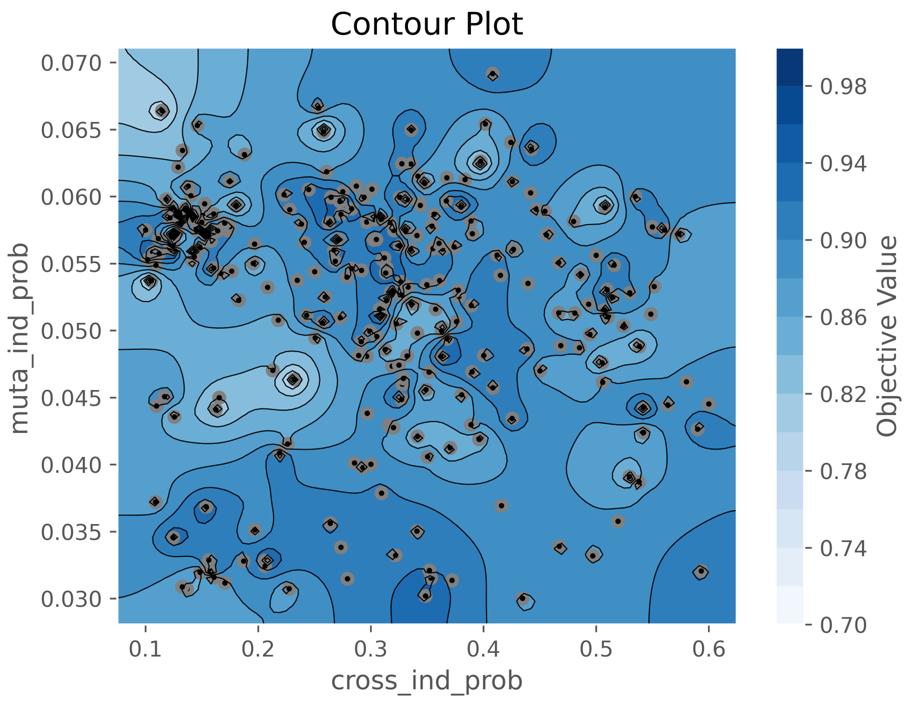


Figure 6: Contour plot of muta_ind_prob vs cross_ind_prob, two hyper parameters of the GA. The objective value is CV score on the classifier that the GA ran on. It's easy to see here that for the hyper-parameters muta_ind_prob and cross_ind_prob the best values are in the square $[0.55, 0.6] \times [0.25, 0.3]$ respectively. These hyper parameters represent the probabilities to independently crate a mutation or a crossover in a single individual.

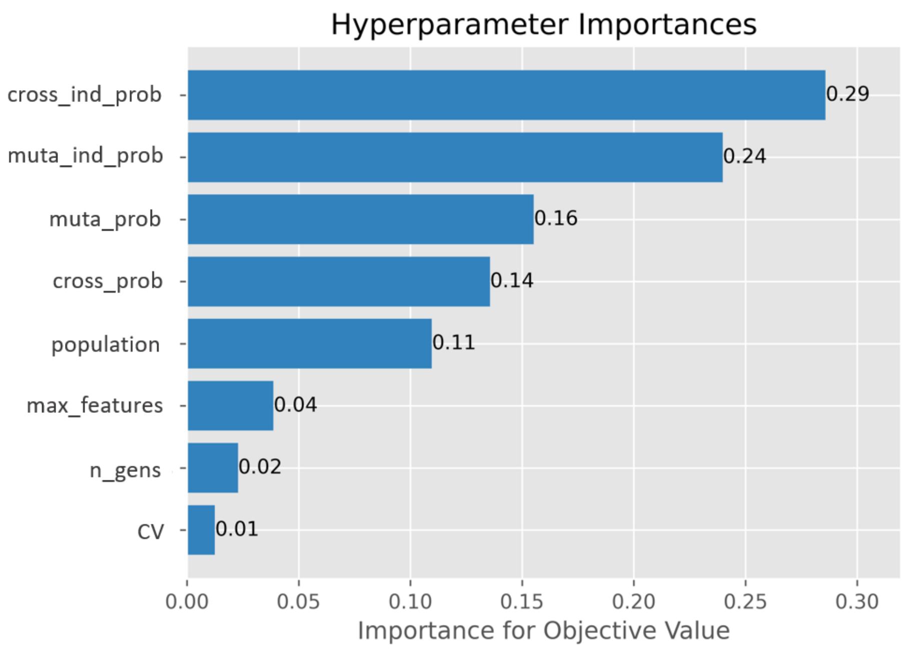


Figure 7: The importance of the parameters ranked according to their effect in the trials.

- Data Cleansing: corrupt data can lead to false conclusion by our classifiers and harm our predictions. In order to prevent this we attempted to remove recording with bad success rates on the basic classifiers before creating the robust classifiers.

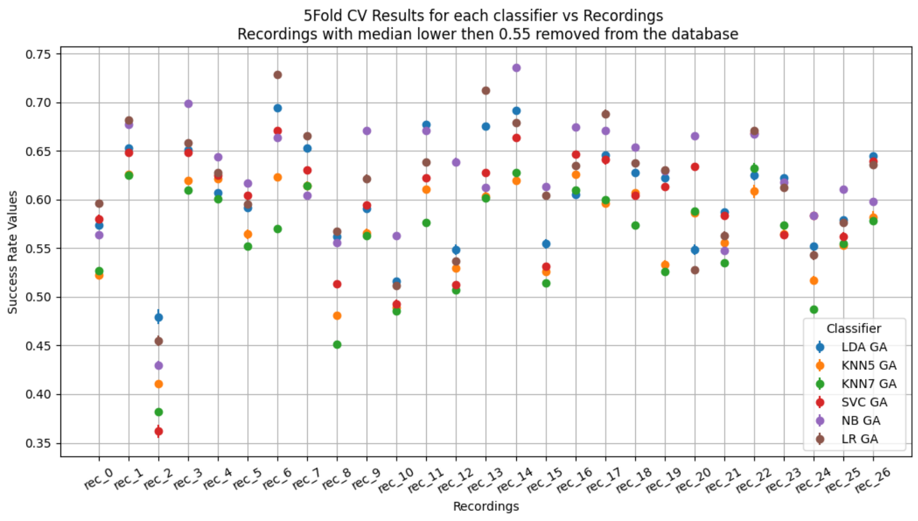


Figure 8: Prediction results over the basic classifiers, vs each recording we hold.

Some of the recordings standout with low success rates over all the classifiers, hence they were removed from the training process.

5 Results

The results that will be shown, are 5-fold CV, using data-sets that each contain 60 samples, including 3 classes (20 samples for each class). We then average the whole data sets (recordings) results, and save this average score for each model. Since GA is a stochastic process, we repeat this process for several times, to get reliable results.

5.1 Using All Classifiers

5.1.1 Initial Results

Let's have a look at the initial results. First we'll have a look on Fig 9

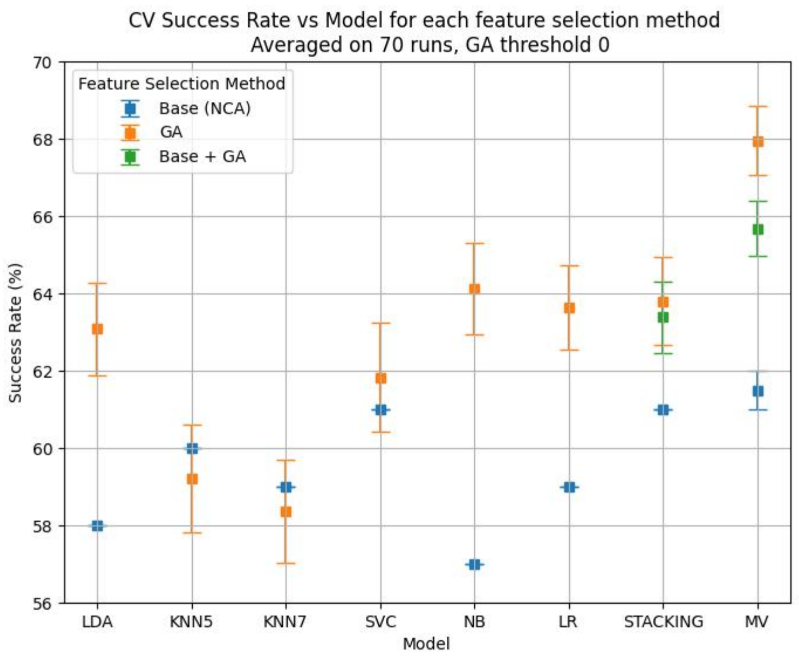


Figure 9: The mean and standard deviation of all classifiers and ensembles, for both the base features and the Genetic algorithm features

From the first observation, it's clear that MV (Majority Vote, i.e. in our case Expert Advice), out performs all the other classifier by a margin of $\sim 4\%$. From looking at MV column, we can conclude that using GA only classifiers yield the best results. This conclusion is relevant to the second ensemble classifier (Stacking). Another interesting observation is that both KNN classifiers performs poorly using the GA selected features. This result could be expected, since using Neighborhood Component Analysis (NCA) feature selection,

returns the optimal features for KNN classifiers.
When we compare stand alone classifiers vs. ensemble classifiers, we can see that Ensemble GA based classifier outperforms single models. We can get the same conclusion when looking only at NCA based classifiers.

5.1.2 Using Thresholds

In order to reduce our feature space we've attempted to limit our system to choose only features that were selected for over a certain threshold in our initial attempt which was comprised of 70 runs. The results cab be seen in figures 10 and 11

At first we've attempted to use only features that were used for over 50 times

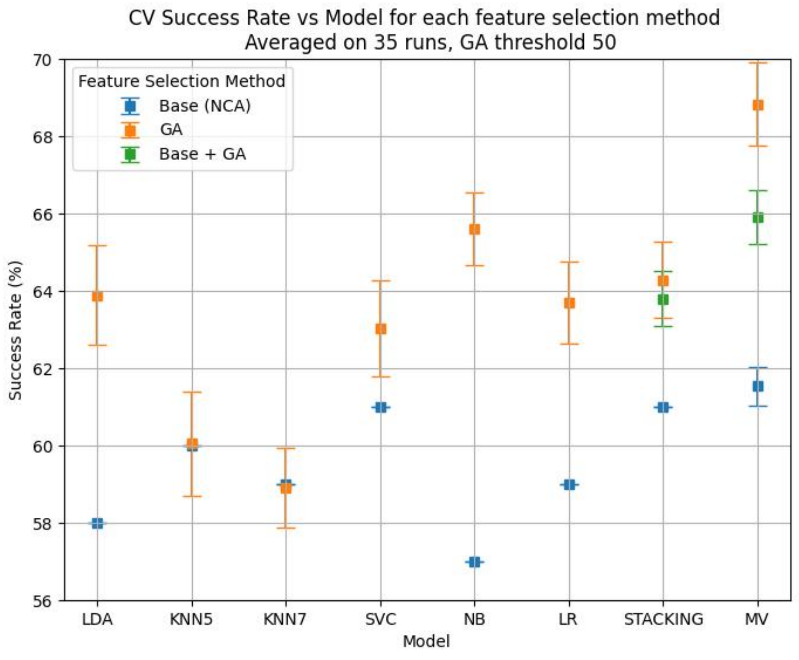


Figure 10: The same figure as figure 9 but only using features that were selected for over 50 times in the first 70 runs

Then, in attempts to reduce the dimension of our feature space even more, we've tried to classify with features that were chosen over 80 times in the initial run.

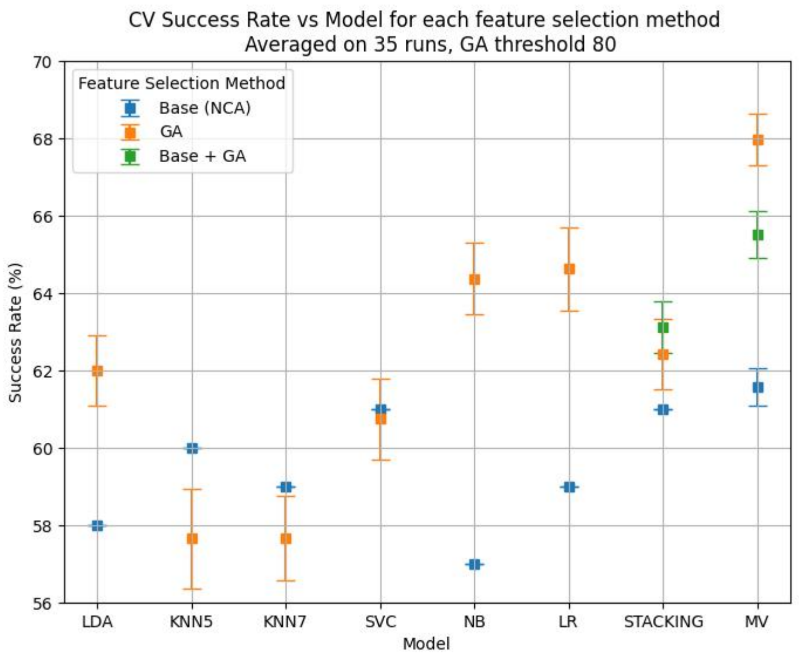


Figure 11: The same figure as figure 9 but only using features that were selected for over 80 times in the first 70 runs

A few observations:

- From Fig 10 we can see a minor improvement in the MV classifier when using only GA based classifiers, and when using Base + GA classifiers. We can observe minimal margin among all the other classifiers.
- From Fig 11 we can observe similar results to the original run, shown in Fig 9 when looking at MV classifier. On the other hand, KNNs, LDA, SVC and Stacking classifiers' success rate dropped in about 2%.
- From all of the above figures, 9, 10 and 11 we can cut a decisive conclusion - KNN classifiers are inferior the the other classifiers, and their presence might hurt our ensembles classifiers.

5.1.3 Conclusions

1. All in all, we can see that our best classifier, MVGA (Majority Vote based on GA features) is not tilted much when using different thresholds.

2. On the other hand, the stand alone classifiers can be better or worse depending on which threshold we assign to them. We can see NB and LR are favoring higher thresholds, unlike SVC, KNNs and LDA, which prefer a better place in the middle.

3. Stacking classifier is not that robust to major changes in its classifiers quality. While we saw MVGA performs well on all thresholds, when comparing to his initial performance, Stacking classifier is being harmed much more, due to the fact that this classifiers is approximately giving similar importance for each classifier prediction, since every base classifier prediction is a dimension in its feature space.

4. Last but not least, even though we are reducing the problem complexity, we are able to maintain the original performance of our classifiers.

5.2 Removing KNN Classifiers

As seen earlier, the KNN classifiers had worse results comparing to the other classifiers. Hence we wanted to know whether their removal would hurt or improve our classifier. In figure 12 we can see very similar results among all classifiers for average over 70 runs with no threshold, implying that the KNN classifiers did not have a significant aid in the predictions of our ensemble classifiers.

After that we've checked the results with the previous thresholds, arriving at the same conclusion that the KNN had little to no impact on our ensemble, as can be seen in figures 13 and 14

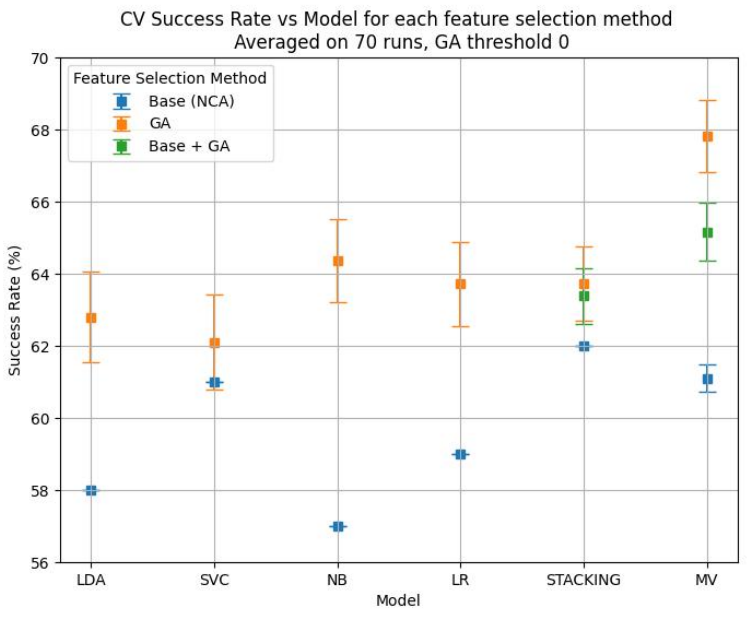


Figure 12: The mean and standard deviation of all classifiers and ensembles, for both the base features and the Genetic algorithm features, this time without KNN classifiers

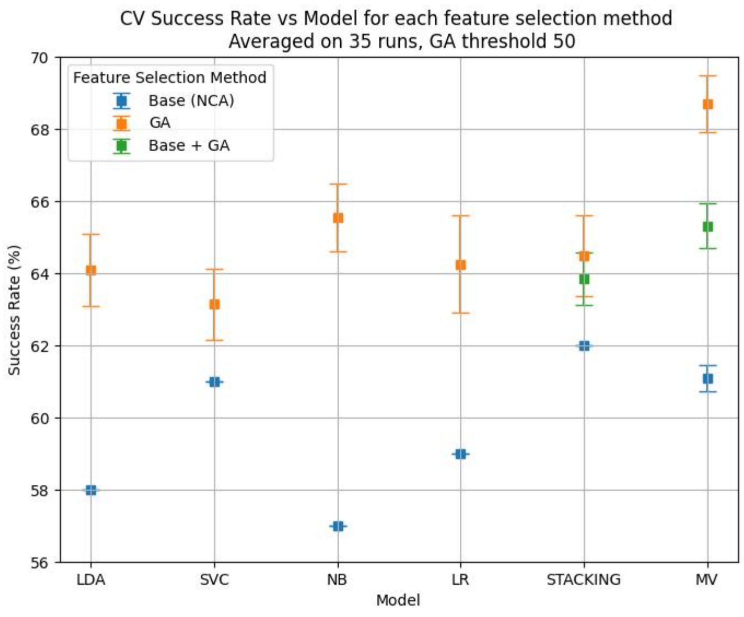


Figure 13: The same figure as figure 12 but only using features that were selected for over 50 times in the first 70 runs

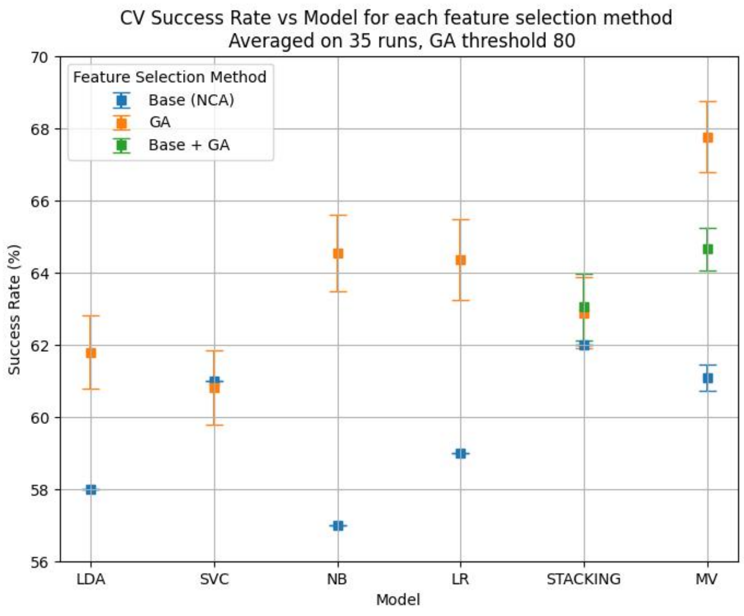


Figure 14: The same figure as figure 12 but only using features that were selected for over 80 times in the first 70 runs

5.2.1 Conclusions

1. Removing KNN classifiers didn't affect much. We get similar score on MVGA classifier, on each threshold with or without KNN as a part of the ensemble.
2. The fact that we get similar performance with/without KNNs using MVGA, is another evidence to a good job that the EA algorithm is doing, and its capability to give little importance to poor classifiers.

5.3 Results Visualization

To better prove our point, we've plotted the MVGA results through all the different configurations that we've shown above.

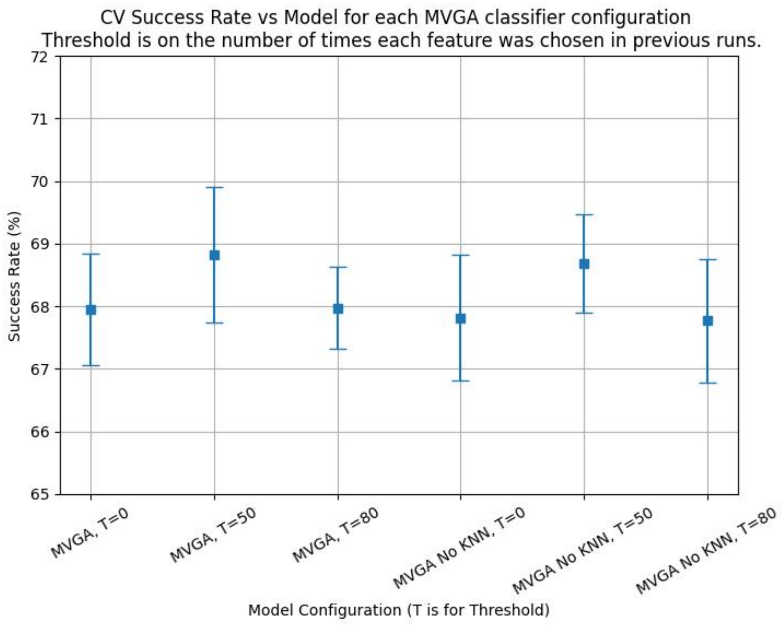


Figure 15: MVGA Avg results and std vs. any configuration we ran on

In fig 15 we can see that through all the configuration, the average score is similar, which provide another evidence to our conclusions.

6 Project Conclusions

6.1 Data-set

A disadvantage the project suffered from is the lack of available data online. Due to the fact that there was no suitable data-set that fitted our needs we spent a lot of precious time building a data-set from scratch. We hope that in the future the data-set we've created will help others as well.

6.2 Statistical Analysis vs Genetic algorithm

The Genetic Algorithm method out preformed the Statistical Analysis by a great margin. A possible reason is the that the data-set wasn't sufficiently large to make statistical conclusions. Another explanation is that evaluation metrics weren't suitable to assess the stationary nature of the data and create a matching feature space, perhaps more research would yield better results.

6.3 Single Classifier vs Ensemble

The Ensemble of several classifiers can mildly improve the results of the system, However it is heavily dependent on the results of the base classifiers. In order to improve the system, one must first improve the base classifiers performance and then adjust the ensemble accordingly. This can be achieved in several ways, for example further adjusting the models hyper-parameters, creating more data to get more accurate results or creating a larger feature space.

6.4 Majority Vote vs Stacking

When comparing the ensemble methods we chose, we got better results from the Majority vote over the stacking method. A possible explanation is that in the MV scheme we favor classifiers with good prediction and are able to improve the weaker prediction of these classifiers, however in the Stacking scheme we regard the prediction of each classifier as a feature and by doing so we give them approximately equal weight, resulting in an average over the classifiers rather than improvement.

6.5 Using Thresholds

When referring to the results we've shown in the previous section, we can tell that using the 50 threshold mark on the feature space, can help us reduce the search problem complexity without degrading our classifiers' scores dramatically, resulting in a much faster GA run time. However after using the 80 threshold mark we can start to see that the quality of the predictions start to deteriorate, putting a limit on the possible threshold we can use to reduce our feature space.

7 Future Work

The work on such project requires multi disciplinary thinking, since we need to first know how to pre process the data, then choose the right features to extract. After all of that is done one needs to choose the right features to be used by the classifier, and choose the right classifier(s).
As can be seen from our work, we mainly concentrated in the subjects of features selection, and classifier selection. Many future works can continue our project:

- Build an online learner using GA selected features. Try to build a classifier that uses the "best" features selected by GA and train him with the chosen features.
- Focus on feature selection: try to use GA to reduce feature space in a "feedback" mode of work. Run few GA iterations, see which features chosen, then choose the most dominant features from statistical point of view, reduce the feature space, and re run few times. This might help reduce feature space, meaning less calculations, hoping for similar/better performance.
- Using GA selection data and NN to find optimal permutation of feature selection.
- Using GA on classifiers - trying to find the best ensemble of classifiers using GA.
- Expand feature space using kernel tricks, then try to reduce the space using the GA feedback method mentioned above.
- Use explain-ability methods to try and mimic GA steps without running GA, and choose the best features according to the explanation.

8 Acknowledgements

We'd like to thank our instructor Ben Kretzu, for giving us research freedom, not tying us down to a given direction, and still giving a helping hand using his experience whenever we needed.

We'd like to thank Koby Kohai and Orly Wigderson from CRML, for the compute resources which were of great help to us, and also for the responsiveness during the making of this project.

We'd like to thank BCI4ALS staff from Ben Gurion University in Beer-Sheva, Or Rabani, Asaf Harel and Lahav Foox, which helped us in ramping in to this project with the Matlab base code [2], and of course supplying us a BCI set which included also technical support.

8.1 Github Link

All of our work can be found on our Github.
We encourage you to make use of project to help in the research for the ALS disease.

References

[1] Sanjeev Arora, Elad Hazan, and Satyen Kale. "The Multiplicative Weights Update Method: a Meta-Algorithm and Applications". In: *Theory of Computing* 8.6 (2012), pp. 121–164. DOI: `10.4086/toc.2012.v008a006`. URL: `https://theoryofcomputing.org/articles/v008a006`.

[2] Asaf Harel. *BCI4ALS-MI*. 2021. URL: `https://github.com/harelasaf/BCI4ALS-MI`.

[3] SK Pahuja, Karan Veer, et al. "Recent approaches on classification and feature extraction of EEG signal: A review". In: *Robotica* 40.1 (2022), pp. 77–101.

[4] Izabela Rejer and Krzysztof Lorenz. "Genetic algorithm and forward method for feature selection in EEG feature space". In: *Journal of Theoretical and Applied Computer Science* 7.2 (2013), pp. 72–82.

[5] David H. Wolpert. "Stacked generalization". In: *Neural Networks* 5.2 (1992), pp. 241–259. ISSN: 0893-6080. DOI: `https://doi.org/10.1016/S0893-6080(05)80023-1`. URL: `https://www.sciencedirect.com/science/article/pii/S0893608005800231`.