Robustness of AI algorithms for neurocomputer interfaces based on software and hardware technologies

Ivan Stefanyshyn[1,*,†], Oleh Pastukh[1,†], Volodymyr Stefanyshyn[1,*,†], Ihor Baran[1,†] and Igor Boyko[1,†]

[1] *Ternopil Ivan Puluj National Technical University, Ruska str., 56, Ternopil, 46001, Ukraine*

Abstract

This article focuses on the use of information technology for brain-computer interaction to detect patterns of brain activity using electroencephalography (EEG) signals. During the experiment, we used machine learning methods, namely the following classifiers: Bagging, Boosting, Nearest Neighbors, and Support Vector. The experiment began with real observations of EEG signals during tasks with finger movements. We used 10-fold cross-validation to evaluate the performance of each classifier, including accuracy and robustness. It was found that the Support Vector classifier showed the highest stability among classifiers. The main goal of the experiment was to determine the importance of robustness of classifiers, especially in medical applications. In conclusion, the experiment contributes to the development of the brain-computer interaction area and the development of robust neuro-interface technologies with practical applications in health care and other places.

Keywords

neuro-interface, artificial intelligence, parallel programming, high-performance computing, robustness, information technology, cloud cluster.

1. Introduction

Today, the world is approaching the gradual improvement of neural interfaces. Therefore, we also want to talk about the potential of using information technologies in medical rehabilitation and diagnostics. For example, research in the brain-computer interface area already makes it possible to read brain activity to predict motor intentions or even to control prostheses. In addition, brain-computer interfaces may have the

CITI'2024: 2nd International Workshop on Computer Information Technologies in Industry 4.0, June 12–14, 2024, Ternopil, Ukraine

* Corresponding author.

† These authors contributed equally.

ivan_stefanyshyn0707@tntu.edu.ua (I. Stefanyshyn); ol_pas@tntu.edu.ua (O. Pastukh); volodymyr_stefanyshyn3006@tntu.edu.ua (V. Stefanyshyn); Ihor.remm@gmail.com (I. Baran); boyko.i.v.theory@gmail.com (I. Boyko)

0009-0008-6930-528X (I. Stefanyshyn); 0000-0002-0080-7053 (O. Pastukh); 0009-0007-2829-8995 (V. Stefanyshyn); 0000-0002-8153-2476 (I. Baran); 0000-0003-2787-1845 (I. Boyko)

potential to improve the process of diagnosing brain disorders such as epilepsy or mental illness.

Along with the significant technological breakthroughs that are occurring in the neural interface area, there is a need to develop an appropriate ethical framework for their use. This is particularly important in the context of data privacy, security, and equitable access to these information technologies.

First of all, it's really important to keep data private when using brain interfaces. This may include personal data such as thoughts, emotions, and even the psychological state of the user. We need to ensure that this information is protected from unauthorized access. This is an extremely important task for developers and users of such technologies.

Second, security and preventing abuse of these interfaces is also a key point. Today we live in a modern world and it is not surprising that neural interfaces can have direct access to control devices or even systems, which requires reliability and protection against abuse by attackers.

Finally, fair access to neural interfaces is important for society. This means that everyone who can take advantage of these information technologies should be able to access them without restrictions, and any barriers should be overcome to ensure equality in their use and enjoyment.

Therefore, the development of neural interfaces is associated not only with technological challenges but also with ethical aspects that require careful attention and the development of appropriate regulatory frameworks.

2. Purpose

We worked hard to improve the robustness of the results during our research. One of the ways to achieve this goal was to apply careful analysis and comparison of different classifiers to decode brain activity. We conducted a wide range of experiments using different machine learning algorithms and signal processing techniques to detect the most effective approaches. We also monitored the level of stability and reproducibility of the results to ensure the robustness of our conclusions.

In addition, we used cloud computing techniques that allowed us to scale our calculations and ensure stable operation of the system even with large amounts of data. This allowed us to increase the accuracy and robustness of our results, as well as reduce the time required for data analysis.

Also, we used parallel programming and high-performance computing to optimize data processing speed and ensure system stability. This allowed us to quickly detect and correct possible errors and increase the robustness of our results.

In summary, our efforts to improve robustness consisted of a combination of full method analysis, using of cloud computing, and parallel programming techniques, and applying of various machine learning algorithms. This allowed us to create a robust and efficient system for decoding brain activity and working with brain-computer interfaces.

3. Presenting the main material

The input data in this article were obtained as a result of real experiments. These experiments are based on advanced methods of researching brain activity using EEG. EEG signals will allow obtaining unique data that help to better understand the internal processes of the brain, such as cognitive functions and motor activity. These findings will become an important basis for the development of advanced neuro-interface technologies and will contribute to the creation of innovative methods of interaction between the brain and the computer.

We used a NEUROKOM computer electroencephalograph to carry out experiments on collecting electroencephalographic (EEG) signals of the brain (Figure 1). This electroencephalograph is equipped with a 16-channel selection of encephalograms, as well as data transfer to a personal computer according to the appropriate protocol. This protocol is the fifth generation of developed computer electroencephalography complexes.

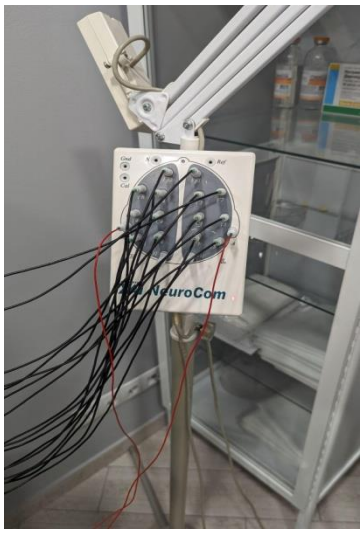

Figure 1: Photo taken during the experiment.

We performed a series of finger movements during EEG data recording experiments. Before starting, each finger was assigned a number from 1 to 5, and the count started with the thumb marked as 1. We performed continuous finger movements including various combinations from 1 to 5. These combinations were carefully designed to ensure consistency and control over them.

We performed only one combination of finger movements during each session. Our goal was to ensure the stability of the experimental conditions and reduce the possibility of unreliability of the results, which in turn would allow us to obtain a more accurate picture of the neural activity during finger movements. The duration of each experimental session was approximately 2 minutes, which ensured that a sufficient amount of data was obtained for further analysis and conclusions.

We received a lot of results, which were related to their corresponding movements, after conducting experiments. Therefore, we want to show how some of them look on the

graphs. To do this, we will display the record number horizontally, and the signal value vertically. There will be a total of 16 curves on the graph, which will display the state of the change in the signal value. We have selected only two experiments for visual presentation. The first image shows the EEG signals recorded during the movements of two fingers, namely the second and third (Figure 2). The second image shows the EEG signals recorded during the movements of only the fourth finger (Figure 3).

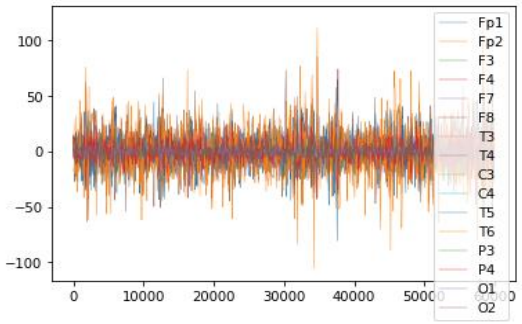

Figure 2: Illustration of EEG signals during movements of second and third fingers

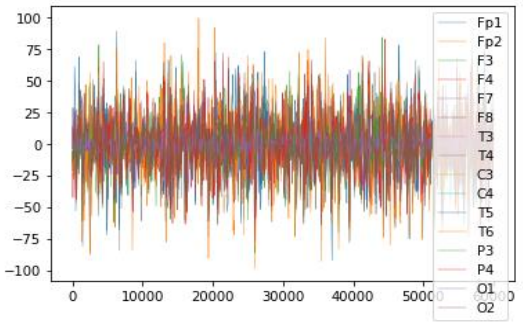

Figure 3: Illustration of EEG signals during movement of fourth finger

The EEG information displayed in the graphs reveals the complex neural connections that are associated with finger movements. Even if these data may seem incoherent to an ordinary observer, modern technologies allow us to process and extract significant useful information from them. Even with all their complexity of understanding, EEG signals provide important and necessary data on neural variability during the performance of motor tasks, as it was conducted by us. This is critical to advancing the development of brain-computer interfaces and our understanding of how the brain works.

4. Explaining of investigation

Now we can look at the data after the experiment. We obtained several files with EEG signals according to the number of finger combinations. Before performing calculations on the received data, we must first prepare them. First of all, we do not feed filter any records, because they are very necessary or even key for further calculations. We removed the column that was responsible for the recording time of the signals. We have replaced this column with a column that contains the number of finger combinations. The new column will help us in future calculations to detect the combination of finger movements from the entire data set. After we have added a new column, we merge all the EEG signal sets. Now we can easily understand which movement a particular entry belongs to because we have previously added a column that contains information about it.

We used machine learning methods to determine finger combinations relative to the input data. Machine learning methods are very well suited for this type of problem. They can easily deal with such a confusing set of data and extract from all this mess which will help us in correctly determining the combination of simultaneous finger movements.

Today, machine learning methods are very popular, and therefore there are now many different classifiers implemented. Each of the classifiers has its advantages and implementation, so we chose the 4 most suitable classifiers to solve the problem of this article. Selected classifiers include Bagging, Boosting, Nearest Neighbors, and Support Vector. As we said, each of these classifiers has its specific features, so they can be ideal for solving a certain type of problem or show poor results when computing other problems. That is why it is necessary to test each of the classifiers in order to obtain the best results. Therefore, we will now provide a brief description of each of the classifiers. Bagging classifier is an ensemble learning method based on the idea of building many independent models that are trained on different subsets of the training data. The results of these models are then aggregated, for example by voting or averaging, to produce a final prediction. Bagging is particularly effective in reducing dispersion and improving the model's robustness to overtraining. Boosting classifier is a method based on the idea of sequentially building a set of weak models that compensate for each other's disadvantages. Each subsequent model focuses on examples where the previous models make mistakes, thus providing a consistent improvement in classification accuracy. The Nearest Neighbors classifier is a simple classification algorithm that detects the class of a record by comparing it to a certain number of nearest records from the training data set. This method is particularly effective for problems with a small number of features and a large amount of training data. The Support Vector classifier is a powerful classification algorithm that partitions the feature space using a hyperplane. He tries to find the optimal hyperplane that maximally separates objects of different classes. This method is effective for both linearly separable and non-linearly separable data sets, thanks to the use of kernel functions.

We need to discuss before we start computing exactly how we will deploy the hardware and software. The software is an implementation of several algorithms according to our chosen classifiers, and it also includes the use of parallel programming to provide high-performance computing. The hardware will be the launch of the software implemented by us on cloud clusters. Also, cloud clusters will allow us to perform our tasks on several clusters at once. This means that we will be able to use all the available resources on the cluster that are available to us to perform the computation as quickly as possible. Now we will consider step by step how it will all be deployed (Figure 4). First of all, we implemented algorithms for each of the classifiers using the Scikit-Learn library. All necessary implementations of classifiers were presented in this library. The next step is to use the Joblib library. This library is the basis of the Scikit-Learn library, so thanks to it, the process of parallelization of tasks is done. The library allowed us to use all possible resources, namely cores and their threads. To enable this feature, we need to pass the n_jobs=-1 hyperparameter. After that, we used the Dask library to improve the distributed computing process and make it really high-performance. This library allowed us to scale the distribution of parallel computations within the available resources in the form of cores and their threads. And finally, we used cloud clusters as a hardware layer. We would like to emphasize that the process of parallelization and high-performance computing takes place on cloud clusters.

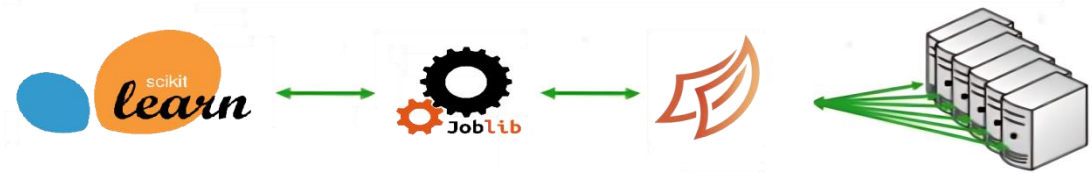

Figure 4: Software – hardware computer calculation pipeline. [Public domain], via Dask. (https://ml.dask.org/joblib.html)

Now we can consider the process of calculating the accuracy of each classifier after a detailed study of them and the methods of their deployment on cloud clusters. We used cross-validation using the cross_val_score function from the sklearn package. This function helped us find the accuracy of each classifier and in further calculations the robustness. Cross-validation involves dividing the data set into subsets and iteratively training the classifier on each of them, evaluating its accuracy on the remaining data. Also, this method accepts different hyperparameters and especially different metrics. We used three metrics, among them accuracy, f1_weighted and roc_auc_ovr_weighted. The cross-validation function takes the number of folds as a hyperparameter, for which we set 10 folds to obtain reliable estimates of the accuracy of each classifier. Another important hyperparameter that we specified was the n_jobs hyperparameter. We previously wrote about how it allowed us to efficiently use the resources of the computing system to perform calculations of different folds in parallel, which increased the efficiency of training and evaluation. Such an approach can provide an opportunity to choose the most suitable model for the analysis of EEG signal data, based on its robustness and performance.

Our next step was to calculate the dispersions for each classifier metric. We calculated the dispersion for each result of the cross-validation function. The dispersion values show how much the classifier differs in its ability to correctly detect the motor combination of fingers. We decided to use the std function from the NumPy library. This function allowed us to programmatically calculate the dispersion, we only needed to pass the resulting accuracy values of each metric to get an instant dispersion result.

Next, we needed to reduce dispersion values for each of the classifiers into one scalar value to make the robustness comparison process more understandable. Therefore, we can represent the resulting tuple of three dispersion values as a vector. From here we can easily calculate the modulus of the vector, which consists of three dispersion values representing different metrics (Formula 1).

$$\text{dispersion} = \sqrt{\text{dispersion\_accuracy}^2 + \text{dispersion\_f1\_weighted}^2 + \text{dispersion\_roc\_auc\_ovr\_weighted}^3} \quad (1)$$

This approach allowed us to obtain a general assessment of the robustness and performance of the classifiers under different cross-validation conditions, giving us an understandable value of their robustness and providing a basis for conclusions regarding the selection of the most effective algorithms for further data analysis.

5. Calculating of dispersion

We will now proceed to the calculations and their results, which will be based on using of parallel programming to ensure high-performance calculations.

5.1. Computation for the Baging classifier

We took the Baging classifier to begin with. We ran this classifier through a 10-fold cross-validation function, throwing default parameters into it. We did this for each metric and the result was a table with accuracy data for each of the 10 folds (Table 1).

Table 1
Accuracy Values for the Bagging Classifier Over 10-fold Cross-Validation

fold	accuracy	f1_weighted	roc_auc_ovr_weighted
1	0.7324564	0.7251272	0.96386176
2	0.77256116	0.77048734	0.96992593
3	0.80727338	0.80602393	0.97599864
4	0.81971311	0.81867569	0.97744289
5	0.81463008	0.81202037	0.97645187
6	0.80745194	0.80576649	0.97440786
7	0.80520207	0.80681292	0.97417689
8	0.80695197	0.80665754	0.97545686
9	0.78188203	0.77989445	0.97018646
10	0.76363312	0.76434375	0.96516058

We also created the normalized confusion matrix (Figure 5). This matrix gives us a better understanding of the possible classification results and helps us understand how well the model recognizes each class. From this matrix, we can conclude that, in general, the accuracy for each class is not the same. We can see that for the classes responsible for the movement of two fingers, the Bagging classifier showed slightly lower results compared to the other classes.

Here we can highlight the following conclusions regarding the obtained data:

- For the accuracy metric, the accuracy variation is from 0.7324564 to 0.81971311, and the dispersion is 0.026407567206818656.
- For the f1_weighted metric, the accuracy variation is from 0.7251272 to 0.81867569, and the dispersion is 0.027785827063570318.
- For the roc_auc_ovr_weighted metric, the accuracy variation is from 0.96386176 to 0.97744289, and the dispersion is 0.004556059924689737.

As a result, the general dispersion value for the Bagging classifier is approximately 0.038602713290993476.

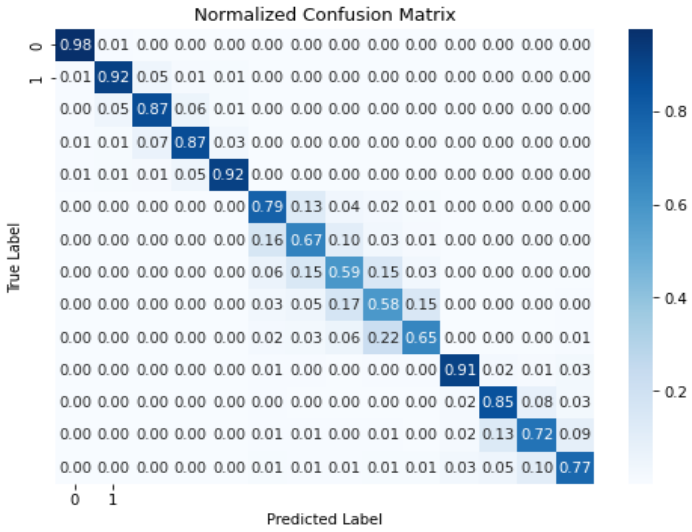

Figure 5: Normalized confusion matrix for the Baging classifier

5.2. Computation for the Nearest Neighbors classifier

We calculated general robustness for bagging, and now we will do it for the Nearest Neighbors classifier. In this case, we also performed 10-fold cross-validation. However, we threw in the hyperparameter n_neighbors=5, which determines the number of neighbors [7]. Below are the tables with the results of the cross-validation (Table 2).

Table 2

Accuracy Values for the Nearest Neighbors Classifier Over 10-fold Cross-Validation

fold	accuracy	f1_weighted	roc_auc_ovr_weighted
1	0.73599191	0.72832798	0.94928696
2	0.77988215	0.77830871	0.95956925
3	0.80229748	0.80123768	0.96431717
4	0.81768942	0.8182214	0.96854029
5	0.8111541	0.81189017	0.9661255
6	0.8008928	0.80125194	0.96337834
7	0.80428546	0.80455047	0.96475618
8	0.79980954	0.79988526	0.96487807
9	0.77350158	0.77325333	0.95735871
10	0.76053806	0.76015201	0.9538284

We also additionally formed the normalized confusion matrix (Figure 6). From this matrix, we again got similar results as with the previous matrix, we got slightly worse accuracy for the movement of two fingers.

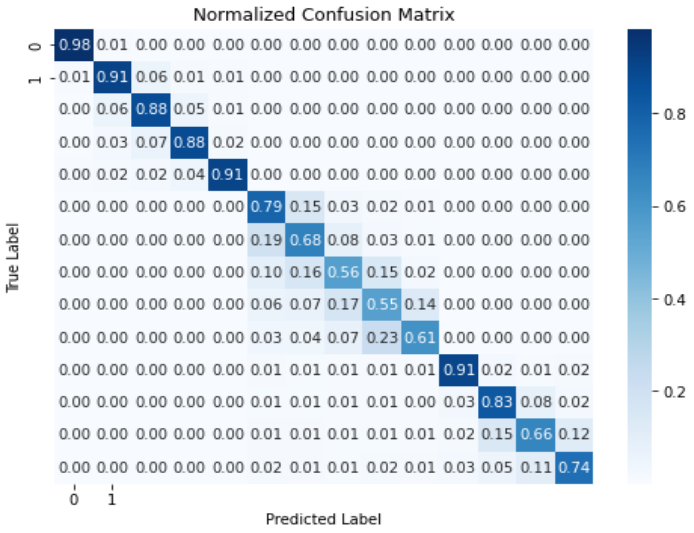

Figure 6: Normalized confusion matrix for the Nearest Neighbors classifier

We can make the following conclusions regarding the data obtained:

- For the accuracy metric, the accuracy variation is from 0.73599191 to 0.81768942, and the dispersion is 0.024339987140229963.
- For the f1_weighted metric, the accuracy variation is from 0.72832798 to 0.8182214, and the dispersion is 0.02626193715472385.
- For the roc_auc_ovr_weighted metric, the accuracy variation is from 0.94928696 to 0.96854029, and the dispersion is 0.005765870693628227.

As a result, the general dispersion for the Nearest Neighbors classifier is approximately 0.03626802423569371.

5.3. Computation for the Boosting classifier

Now we move on to the next classifier called Boosting. As with the previous classifiers, we also performed a 10-fold cross-validation. It is worth reminding that we threw several hyperparameters into this classifier. The first hyperparameter was n_estimators=100, which means the number of boosting stages to perform. The second hyperparameter learning_rate=1.0 affects the learning rate. The third hyperparameter max_depth=3 specifies the depth of individual regression estimators. Oh, and the last hyperparameter random_state=0, which affects the random seed given to each Tree estimator at each boosting iteration [7]. After that, we performed cross-validation for each of the metrics and recorded the results in the table below (Table 3).

Table 3
Accuracy Values for the Boosting Classifier Over 10-fold Cross-Validation

fold	accuracy	f1_weighted	roc_auc_ovr_weighted
1	0.08919707	0.07135361	0.50956754
2	0.11818939	0.1021653	0.52517873
3	0.12928992	0.12465008	0.53115581
4	0.1995536	0.19780656	0.56985162
5	0.12019523	0.09215438	0.52625847
6	0.09452345	0.07986294	0.51763429
7	0.12345678	0.11245725	0.55511208
8	0.10567891	0.13329547	0.52789746
9	0.13131313	0.18820467	0.56124519
10	0.09786543	0.08573129	0.54482376

We also created the normalized confusion matrix (Figure 7). Here we can see that the accuracy for each class is very small in contrast to the previous matrixes.

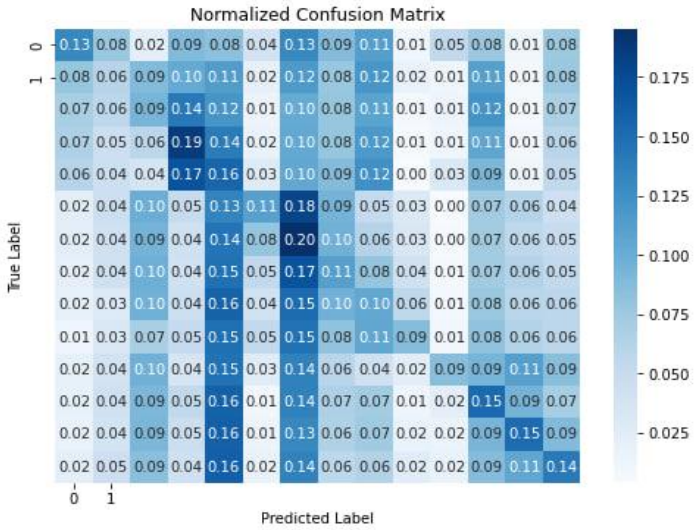

Figure 7: Normalized confusion matrix for the Boosting classifier

We can make the following conclusions regarding the data obtained:

- For the accuracy metric, the accuracy variation is from 0.08919707 to 0.1995536, and the dispersion is 0.029736607577950602.
- For the f1_weighted metric, the accuracy variation is from 0.07135361 to 0.19780656, and the dispersion is 0.041462155780550354.
- For the roc_auc_ovr_weighted metric, the accuracy variation is from 0.50956754 to 0.56985162, and the dispersion is 0.018856984196290352.

As a result, the general dispersion for the Boosting classifier is approximately 0.054396342204184996.

5.4. Computation for the Support Vector classifier

Finally, we will consider the Support Vector classifier. Again, we performed cross-validations with 10-fold for this classifier. In this classifier, we threw hyperparameter gamma=auto, which is a kernel coefficient and in this case, it tells the classifier to use 1 or the n_features attribute. Below we have displayed the cross-validation results for each of the metrics (Table 4).

Table 4

Accuracy Values for the Support Vector Classifier Over 10-fold Cross-Validation

fold	accuracy	f1_weighted	roc_auc_ovr_weighted
1	0.75016103	0.76299484	0.98311633
2	0.82829593	0.84031427	0.96719612
3	0.82829593	0.82274713	0.95916282
4	0.82829593	0.8131163	0.96527478
5	0.74046783	0.75016103	0.97192364
6	0.76457291	0.75548759	0.97857249
7	0.75919182	0.78123617	0.96208537
8	0.77282748	0.79016348	0.97610152
9	0.75736927	0.81299358	0.96676829
10	0.77704019	0.80746326	0.97341714

We created the normalized confusion matrix (Figure 8). From this matrix, we can conclude that the accuracy of detecting the movement with one finger is higher than that of other combinations.

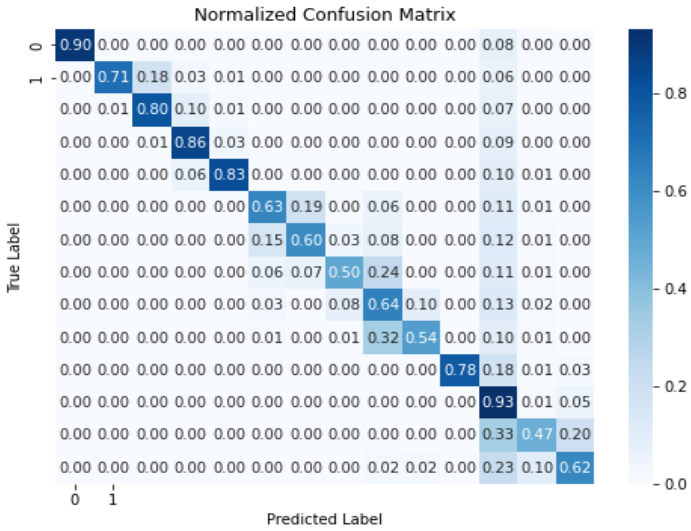

Figure 8: Normalized confusion matrix for the Support Vector classifier

We can make the following conclusions regarding the data obtained:

- For the accuracy metric, the accuracy variation is from 0.74046783 to 0.82829593, and the dispersion is 0.032695151367140296.
- For the f1_weighted metric, the accuracy variation is from 0.75016103 to 0.84031427, and the dispersion is 0.029020284057750808.
- For the roc_auc_ovr_weighted metric, the accuracy variation is from 50956754 to 56985162, and the dispersion is 0.007194034759436455.

As a result, the general dispersion for the Support Vectors classifier is approximately 0.04430467182851874.

Conclusions

After calculation, we obtained the general dispersion for each classifier. From here we can make the following conclusions: the Nearest Neighbors classifier showed the lowest dispersion (0.0363), which indicates its highest robustness and the lowest spread in the measured metrics. This means that it shows more stable and reliable results compared to other classifiers. The Bagging classifier is next in terms of robustness, which also showed a fairly low level of dispersion (0.0386), but slightly higher than that of the Nearest Neighbors classifier. However, the difference is very small. The Support Vector classifier showed significantly higher dispersion (0.0443), which may indicate less stability of its results compared to other classifiers. Finally, the Boosting classifier showed the highest dispersion (0.0544), which may indicate the largest spread of results and the least robustness compared to the other classifiers. Moreover, the accuracy of this classifier in this experiment was extremely low. As a result, we concluded that to ensure the best robustness for solving the task of recognizing finger movements from EEG signal data, we recommend using Nearest Neighbors or Bagging classifiers from the four classifiers that we proposed, given the critical importance of reliability in measurements of medical data.

Also, we especially recommend keeping in mind that the robustness and reliability of the classifier are particularly important for applications in medical research and practice. For example, for the development of information technologies such as brain interface systems, where the accuracy and robustness of brain signal classification detect the effectiveness and safety of using such systems in patients with neurological diseases. From these results, it can be concluded that to choose the optimal classifier for a specific task, it is important to consider both its performance and the robustness of the results, especially in medical data measurements, where robustness is critical.

References

[1] Musk E., Neuralink. An Integrated Brain-Machine Interface Platform With Thousands of Channels. Journal of medical Internet research. 2019. Vol. 21. No. 10. URL: https://doi.org/10.2196/16194.

[2] Gilja, V., Pandarinath, C., Blabe, C. H., Nuyujukian, P., Simeral, J. D., Sarma, A. A., ... & Henderson, J. M. Clinical translation of a high-performance neural prosthesis. Nature Medicine. 2015. URL: https://pubmed.ncbi.nlm.nih.gov/26413781.
[3] Ajiboye, A. B., Willett, F. R., Young, D. R., Memberg, W. D., Murphy, B. A., Miller, J. P., ... & Walter, B. L. Restoration of reaching and grasping movements through brain-controlled muscle stimulation in a person with tetraplegia: a proof-of-concept demonstration. 2017. URL: https://pubmed.ncbi.nlm.nih.gov/28363483.
[4] Pastukh O., Stefanyshyn V., Baran I., Yakymenko I., Vasylkiv V. Mathematics and software for controlling mobile software devices based on brain activity signals. The International Workshop on Information Technologies: Theoretical and Applied Problems (ITTAP-2023): Proceedings of the International Scientific and Practical Conference, Ternopil, November 22-24, 2023. pp. 684 - 689.
[5] Medic XAI. URL: https://xai-medica.com/en/equipments.html.
[6] Scikit-Learn & Joblib. URL: https://ml.dask.org/joblib.html.
[7] Scikit-Learn. URL: https://scikit-learn.org/.
[8] Bright brain – London's eeg, neurofeedback and brain stimylation centre. URL: https://www.brightbraincentre.co.uk/electroencephalogram-eeg-brainwaves.
[9] Zuo C., Jin J., Yin E., Saab R., Miao Y., Wang X., et al. Novel hybrid brain-computer interface system based on motor imagery and p300. Cogn. Neurodyn. 2020. Vol. 15. No. 2. URL: https://doi.org/10.1007/s11571-019-09560-x.
[10] Software – hardware computer calculation pipeline. URL: https://ml.dask.org/joblib.html.