Energy-aware KNN for EEG Classification: A Case Study in Heterogeneous Platforms

Juan José Escobar[1][0000−0002−4258−0264], Francisco Rodríguez[2], Rukiye Savran Kızıltepe[3][0000−0002−3862−7621], Beatriz Prieto[2][0000−0003−0187−6533], Dragi Kimovski[4][0000−0001−5933−3246], Andrés Ortiz[5][0000−0003−2690−1926], and Miguel Damas[2][0000−0003−2599−8076]

[1] Department of Software Engineering, CITIC, University of Granada, Spain
jjescobar@ugr.es
[2] Department of Computer Engineering, Automation, and Robotics, CITIC, University of Granada, Spain
cazz@correo.ugr.es
{beap,mdamas}@ugr.es
[3] Sofware Engineering, Karadeniz Technical University, Turkey
rukiye.savrankiziltepe@ktu.edu.tr
[4] Institute of Information Technology, University of Klagenfurt, Austria
Dragi.Kimovski@aau.at
[5] Department of Communications Engineering, University of Málaga, Spain
aortiz@ic.uma.es

Abstract. The growing energy consumption caused by IT is forcing application developers to consider energy efficiency as one of the fundamental design parameters. This parameter acquires great relevance in HPC systems when running artificial neural networks and Machine Learning applications. Thus, this article shows an example of how to estimate and consider energy consumption in a real case of EEG classification. An efficient and distributed implementation of the KNN algorithm that uses mRMR as a feature selection technique to reduce the dimensionality of the dataset is proposed. The performance of three different workload distributions is analyzed to identify which one is more suitable according to the experimental conditions. The proposed approach outperforms the classification results obtained by previous works. It achieves an accuracy rate of 88.8% and a speedup of 74.53 when running on a multi-node heterogeneous cluster, consuming only 13.38% of the energy of the sequential version.

Keywords: Parallel and distributed programming · Heterogeneous clusters · Energy-aware computing · EEG classification · KNN · mRMR.

1 Introduction

Greenhouse gas emissions caused by energy consumption associated with the proliferation of equipment, applications, and computer programs is growing worryingly. There are estimates that determine that ICT could contribute up to

23% of global greenhouse gas emissions in 2030 [8]. To reduce consumption, it is necessary to address the problem simultaneously from different approaches. One of them is to analyze the energy consumption of programs and try to run them in the most energy-efficient configuration. The other one, to exploit the qualities of distributed and heterogeneous parallel platforms. Currently, when designing applications, it is not enough to consider only the precision of the results and execution time, but energy efficiency must also be considered as a fundamental parameter. Among the most complex applications in terms of execution time and energy consumption are those related to Machine Learning. These High-Performance Computing (HPC) applications often have to process large amounts of data (Big Data) and are characterized by high algorithmic complexity. Therefore, this work shows the results of an investigation on the energy efficiency of a bioengineering application, based on the KNN (K-Nearest Neighbors) algorithm, capable of exploiting the qualities of a distributed and heterogeneous parallel platform, which also takes into account performance in terms of accuracy of results and execution time.

After this introduction, the rest of the article is structured as follows: Section 2 refers to different works in the literature related to the topic addressed. Section 3 details the proposed KNN implementation for EEG classification. Then, Section 4 analyzes the experimental results and discusses the importance of energy awareness in HPC systems. Finally, Section 5 provides the conclusions.

2 Background

The use of artificial neural networks and Machine Learning techniques in bioinformatics has experienced exponential growth in recent years. Among other causes is the considerable growth in the size of biological datasets, as in the case of Electroencephalography. Bioinformatics, among other topics, deals with EEG signals, which represent the electrical activity of different parts of the brain. EEG signals are used to aid in the diagnosis of disorders such as schizophrenia [1], epilepsy [4], dyslexia [19], depression [17], autism [11], or sleep problems [18]. They are also used to classify motor functions of the nervous system, such as movement of limbs or eyes [16], and for the classification of human emotions [14]. However, the main problem of working with EEG signals is their high dimensionality, which makes their correct classification difficult since most of them do not contain relevant information. Therefore, it is important to apply feature selection techniques to obtain the most relevant ones. One of the fundamental objectives of the artificial neural networks and Machine Learning is to recognize patterns in these signals for their subsequent classification. Indeed, the EEG classification problem is usually addressed through various Machine Learning methods, such as regression or clustering algorithms.

In this work, a KNN algorithm for instance-based supervised classification has been considered due to its good performance in this type of applications. The KNN algorithm generally tries to classify the instances (patterns) by assigning

them to the predominant class among their K nearest neighbors. The steps required to classify each new instance are:

1. Calculate the distance between the instance to classify and the rest of training samples. In this work, the Euclidean distance has been used.
2. Sort the distances in increasing order.
3. Identify the predominant class among the nearest K distances (neighbors).
4. Assign the new instance to the predominant class.

3 The Proposed KNN for EEG Classification

EEG classification has been approached using the KNN algorithm together with the minimum Redundancy Maximum Relevance (mRMR) technique [12], which sorts the N_F features of the dataset from least to most relevant. This approach helps to deal with the dimensionality problem [15] and to reduce computation time by avoiding the evaluation of all 2^{N_F} possible subsets of features. Instead,

Algorithm 1: Pseudocode of the approach used by the workers to evaluate all possible values of K for a feature subset.

```
1  Function evaluateFeatureSubset(Tr, Te, Idx)
       Input  : Training dataset, Tr
       Input  : Test dataset, Te
       Input  : Index of the last column of the feature subset to evaluate, Idx
       Output: Accuracy of the best value of K, acc
2      N_I ← getNumberInstances(Te)
       // Vector with the correct predictions for each value of K
3      C_P ← {0}
       // Set as many OpenMP threads as logical CPU cores
4      #pragma omp parallel for
5      for i ← 1 to N_I test rows do
           // Distance between instance te[i] and all training instances
6          D ← calculateDistances(Te[i], Tr, Idx)
7          for k ← 1 to N_I do
8              P_C ← predominantClass(k, D)
9              if prediction P_C is correct then
10                 C_P[k]++
11             end
12         end
13     end
14     C_P ← sort(C_P, "Descending")
15     acc ← C_P[1] / N_I
16     return acc
17 End
```

the proposed algorithm only evaluates N_F of them, where in each one, the next feature from the list provided by mRMR is added. However, for each subset, all possible values of the K parameter are also evaluated to get the best classification accuracy. The mRMR implementation is based on the one proposed in [5] and takes the F-test Correlation Quotient (FCQ) criteria to select the next feature.

KNN algorithm has been parallelized by distributing feature subsets among worker nodes with MPI library and distributing the test instances to classify among CPU threads with OpenMP. The latter can be seen in the `#pragma omp parallel for` directive of Algorithm 1 (Line 4). The evaluation of all values of K for a test instance has been optimized by calculating its distance from all training instances (Line 6). In this way, the array D is reused in the loop of Line 7 to obtain the predominant class according to the value of K (Line 8). If the prediction is correct, the value of the k-th position of the prediction vector, C_P, is incremented by one. Once all instances have been classified, the prediction vector is sorted and the first position is used to calculate the classification accuracy of the best K (Lines 14 and 15).

3.1 A Distributed Master-worker Scheme for Node-level Parallelism

The proposed implementation, whose pseudocode can be found in Algorithm 2, follows a master-worker scheme where the master tells each worker which feature subset must use to evaluate a KNN. The execution ends when there is no more work to process and the function returns the best accuracy found (Line 36). The operation is as follows: the master waits in Line 8 for some worker to request its first job or to return the result of one of them, which is also implicitly associated with the assignment of a new job. The message type is identified by the MPI tags described in Table 1. When the master receives a result, it checks if the accuracy of that job (feature subset) is better than the current one. If so, update its value (Lines 9 to 11) and send a new chunk with the `JOB_DATA` tag (Line 14). Before sending work to a worker, the master checks for unprocessed chunks. If there is no availability, the worker will receive the `STOP` tag and stop its execution since there is no more work to do (Line 16).

Regarding the workers (Lines 20 to 35), they apply the mRMR algorithm on the training dataset to obtain the ranked list of features. A worker requests jobs by sending a message with the `FIRST_JOB` tag (Line 23). For each chunk of

Table 1. MPI tags used during communications between master and workers.

MPI tag	Description	Sender	Receiver
`FIRST_JOB`	Request for the first job	Worker	Master
`JOB_DATA`	There is work to do	Master	Worker
`RESULT`	Return the result of the job	Worker	Master
`STOP`	No more work to be done	Master	Worker

Algorithm 2: Pseudocode of the master-worker approach.

```
1  Function Main(Tr, Te, C_S, N_Wk)
       Input  : Training and test datasets, Tr and Te
       Input  : Maximum number of features to send to workers (chunk size), C_S
       Input  : Number of workers nodes, N_Wk
       Output: Accuracy of the best feature subset, bestAcc
2      bestAcc ← −1
3      if Master then
4          N_F ← getNumberFeatures(Tr)
5          indexList ← {1, ..., N_F}
6          stops ← 0
7          while stops ≠ N_Wk do
8              MPI::Recv(acc, tag)
9              if tag is RESULT and acc > bestAcc then
10                 bestAcc ← acc
11             end
               // Next chunk to send, e.g. indices 10,11,12 when C_S = 3
12             chunk ← getNextFeatureChunk(indexList, C_S)
13             if size(chunk) > 0 then
14                 MPI::Send(chunk, JOB_DATA)
15             else
16                 MPI::Send(NULL, STOP)
17                 stops ← stops + 1
18             end
19         end
20     else
           // mRMR. Reordering of datasets for coalesced memory access
21         rankedFeatures ← mRMR(Tr)
22         Tr, Te ← sortDatasets(Tr, Te, rankedFeatures)
           // Start asking the master for workloads
23         MPI::Send(NULL, FIRST_JOB)
24         MPI::Recv(chunk, tag)
25         while tag is JOB_DATA do
26             for i ← 1 to size(chunk) do
27                 acc ← evaluateFeatureSubset(Tr, Te, chunk[i])
28                 if acc > bestAcc then
29                     bestAcc ← acc
30                 end
31             end
32             MPI::Send(bestAcc, RESULT)
33             MPI::Recv(chunk, tag)
34         end
35     end
36     return bestAcc
37 End
```

(a) Contiguous blocks (b) Strided blocks

Fig. 1. The two different static workload distributions used by the master node.

features received (Lines 26 to 31), the worker obtains the accuracy of the corresponding feature subset by calling in Line 27 to the `evaluateFeatureSubset` function of Algorithm 1. If the accuracy of the processed feature subset is greater than the existing one, it will be update. Once all the possible subsets of the received chunk have been processed, the worker returns the best accuracy obtained to the master by sending a message with the `RESULT` tag, and waits for the assignment of a new job (Lines 32 and 33). This process is repeated until the `STOP` tag is received, indicating that the worker can end its execution.

3.2 Ways of Distributing the Workload

As previously seen in Section 3.1, feature chunks are sent to workers via the message-passing interface provided by the MPI library. This allows the application to distribute the workload among the different nodes of the cluster. However, in the algorithm proposed here, the workload of each feature subset is asymmetric since the number of features in each one is variable. For example, suppose a dataset with ten features, two worker nodes, and a chunk size of 2. In this scenario, the first chunk that the master will send contains the indices 1 and 2, corresponding to the subsets $\{1\}$ and $\{1, 2\}$. The second worker will receive indices 3 and 4 to compute the subsets $\{1, 2, 3\}$ and $\{1, 2, 3, 4\}$. In other words, a higher index implies computing more features within the KNN and, consequently, a longer execution time. To deal with workload imbalance, by default, the procedure distributes chunks dynamically according to the specified chunk size. Although this has the disadvantage of increasing communications, it is essential in heterogeneous systems to avoid performance drops. If the user wants, the master can also give each worker a chunk of features at the start of the algorithm by dividing the number of total chunks by the number of workers. This can be done in two ways: contiguous or striding features (see Figure 1). The strided assignment could reduce the workload imbalance [6] present in the contiguous blocks alternative since each node would compute similar subsets.

4 Experimental Results and Discussion

All experiments are repeated ten times to obtain more reliable measurements of the application's behavior. The application has been executed in an HPC cluster composed of eight heterogeneous NUMA nodes whose CPU devices are

Table 2. Characteristics of the cluster used in the experiments.

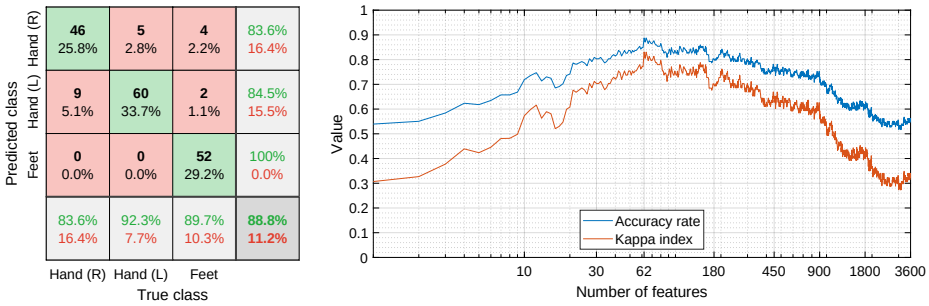
Node	CPU				RAM	
	Model	Total cores/threads	TDP (W)	Frequency (MHz)	Frequency (MHz)	Size (GB)
Master	2x Intel Xeon E5-2620 v2	12/24	160	2,100	1,600	32
1	1x Intel Xeon E5-2620 v4	8/16	85		2,133	
2	2x Intel Xeon E5-2620 v4	16/32	170			
3 to 7	2x Intel Xeon Silver 4214	24/48	170	2,200	2,933	64

detailed in Table 2. The cluster runs the *Rocky Linux* distribution (v8.5) and schedules the jobs using the SLURM task manager (v20.11.7) [10]. The *C++* source codes have been compiled with the GNU compiler (GCC v8.5.0), the OpenMPI library (v4.0.5), and optimization flags `-O2 -funroll-loops`. The energy measurements of each node have been obtained from a custom wattmeter, called *Vampire*, capable of capturing in real-time information of instantaneous power (W) and accumulated energy consumed (W · h) for each computing node.

The EEG dataset belongs to the BCI laboratory of the University of Essex and corresponds to a human subject coded as #104 [3]. The dataset includes 178 signals for training and another 178 for testing, each with 3,600 features. As the signals can belong to three different motor imagery movements (left hand, right hand, and feet), the KNN algorithm deals with a 3-class classification problem.

4.1 Classification Analysis

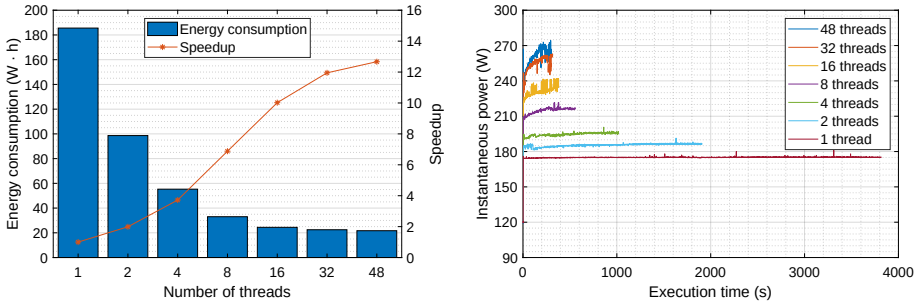
The proposed algorithm achieves a Kappa index of 0.83 using the first 62 features of mRMR and $K = 18$. This solution widely outperforms a run without mRMR (0.34), and other approaches in the literature that use the same dataset: [2] (0.63), [9] (0.70), and [13] (0.76). The result has been validated by replicating its value when executing the KNN with the Python and Matlab languages and the same input parameters. Figure 2(a) shows the corresponding confusion matrix, which reveals an overall accuracy rate of 88.8%. The evolution of the accuracy rate and the Kappa index depending on the number of selected features can be observed in Figure 2(b). The general trend is that both metrics increase as new features are added until reaching the peak (62), and then progressively fall. It seems that the algorithm's convergence is penalized with the selection of many features, which are also irrelevant. It is also observed that the values of accuracy and Kappa distance themselves for extreme values of the graph.



(a) Confusion matrix of the best case (62 features)

(b) Accuracy rate and Kappa index when adding features

Fig. 2. Classification results of the proposed approach when using mRMR and $K = 18$.



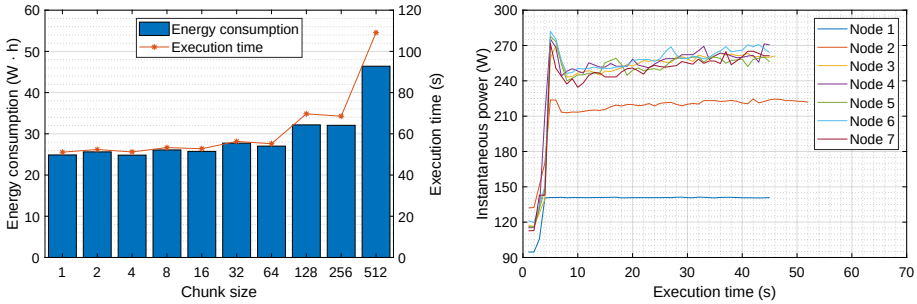
(a) Speedup and energy consumption

(b) Instantaneous power

Fig. 3. Performance of the proposed approach in a single-node configuration when varying the number of OpenMP threads.

4.2 Energy-time Performance

Figure 3 shows the application's performance after running on Node 3. The goal is to depict the speedup scalability of the first parallelism level, which occurs within each computing node when the number of logical CPU cores is increased. From Figure 3(a), it can be seen that the maximum speedup of 12.67 is obtained using the 48 threads available in the node. Its behavior is approximately linear, up to four threads, and logarithmic for higher values. The main reason is that the motherboard supports quad-channel memory. Increasing the number of threads above four causes competition for memory accesses since not all of them can do so simultaneously. It is also due, although to a lesser extent, because the workload for each thread decreases and the cost of managing threads becomes important. This means that the speed gain could increase with larger datasets that allow threads to compute for longer periods. It has also been found that distributing the instances to be classified among the threads statically provides the best performance (Line 4 of Algorithm 1). This is expected, as indicated



(a) Execution time and energy consumption when the chunk size is increased

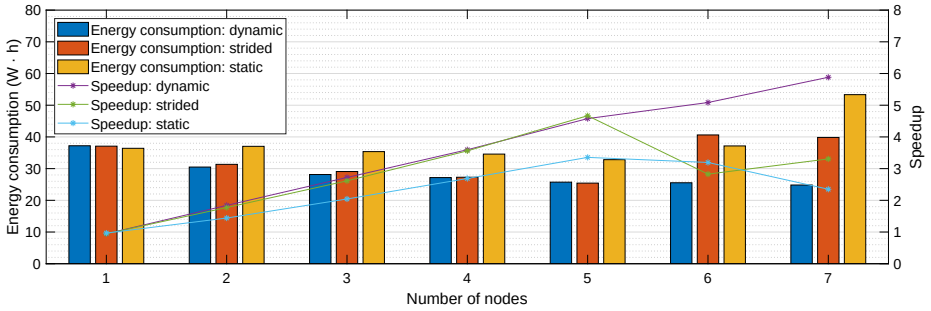
(b) Instantaneous power of the best chunk size found (4)

Fig. 4. Performance of the dynamic workload distribution when using all nodes.

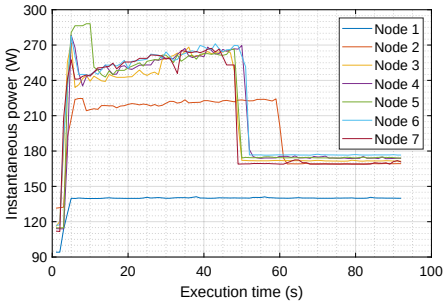
in [7], because the computational workload is the same for each thread, so a dynamic distribution has been discarded. Regarding energy consumption, also for the case of using 48 threads, it provides the lowest total energy consumption. This may seem contradictory since the use of more resources is associated with a higher instantaneous power (see Figure 3(b)). However, energy consumption also depends linearly on execution time, and since speedup increases at a greater rate than energy, total energy consumption is less.

The performance of the hybrid MPI-OpenMP approach that corresponds to the second level of parallelism is shown in Figures 4 and 5. On the one hand, Figure 4 exposes the behavior of the application when all nodes are used, and the workload distribution is dynamic. Figure 4(a) reveals that a very large chunk size leads to worse execution time and energy consumption mainly due to workload imbalance. The instantaneous power of each node for a chunk size of 4 is plotted in Figure 4(b). Despite the fact that the optimal size ranges from 1 to 64, the value 4 has been set as definitive since it works well with few nodes and should do so with more than 7. What is observed in the figure is that most nodes end up simultaneously, which is expected in dynamic workload distributions.

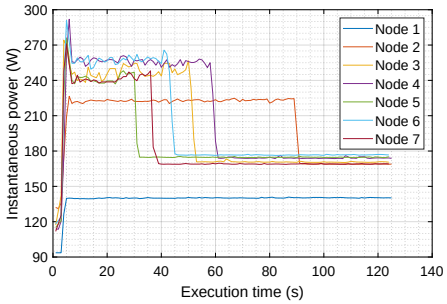
On the other hand, Figure 5 compares the different workload distributions. The number of computation nodes indicated in Figure 5(a) does not correspond to the order shown in Table 2. Instead, the nodes in the graph correspond to homogeneous and heterogeneous ones in that order. That is, first Nodes 3 to 7, and later Nodes 1 and 2. In this way, the scalability of the program can be analyzed according to the type of node added. As expected, for all distributions, the observed speedup grows linearly as more nodes are used, but up to 5 and in different magnitudes. From this point on, only dynamic distribution continues to scale its performance, although to a lesser extent since heterogeneous nodes begin to be used. In fact, it can be seen that the increase in speedup is in line with the added heterogeneous node: adding Node 2 boosts speed up more than adding Node 1 (the slowest one), until reaching a maximum speedup of 5.88. With respect to a sequential execution (1 thread), the application achieves a



(a) Speedup and energy consumption when increasing the number of computing nodes



(b) Instantaneous power of the strided distribution when using all nodes



(c) Instantaneous power of the static distribution when using all nodes

Fig. 5. Comparison of performance between the different workload distributions.

speedup of 74.53, consuming only 13.38% of energy. The use of heterogeneous nodes also negatively affects static and dynamic distributions but in different ways. In the case of strided, speedup plummets for 6 nodes and improves slightly after adding the last one. Not so for the static distribution, which worsens its performance for each node added. The instantaneous power in Figure 5(c) reveals that workload imbalance is responsible. Here, the nodes finish computing in a staggered manner, with a long interval between the first ($t = 30$) and the last ($t = 125$). In the strided case (Figure 5(b)), only the homogeneous nodes finish at the same time, but before the heterogeneous nodes, causing a bottleneck. Based on the results, it can be affirmed that the dynamic distribution provides the best results in speedup, energy consumption, and scalability since the speed gain is very close to the number of nodes used to compute.

5 Conclusions

This work has proposed to investigate the energy efficiency of a bioengineering application capable of exploiting the qualities of distributed and heterogeneous parallel platforms. The use of mRMR for the selection of features has allowed

for the improvement of the performance of existing approaches in the literature that use the same dataset. Another contribution of this article has been to consider energy efficiency as a fundamental parameter, unlike other works that focus only on the accuracy of the results and on the execution time. In addition, different workload distributions for the proposed procedure have been analyzed. The results have verified that a dynamic distribution is the most appropriate option to distribute asymmetric jobs in heterogeneous systems. They have also shown the importance of knowing the architecture when writing parallel code to take advantage of all available resources, reaching speedups of up to 74.53 consuming only 13.38% energy of sequential execution. Even so, the next step is to improve this result using accelerators such as GPUs and increasing data parallelism through vectorization techniques. Another way to reduce energy consumption could be through the use of an energy policy that stops or resumes the execution of the program according to the cost per Megawatt. Consequently, this policy would allow data centers to save energy or money, depending on the user's preferences, but in the latter it would be at the cost of lengthening the execution time and energy consumed.

Acknowledgements

Work funded by the Spanish Ministry of Science, Innovation, and Universities under grants PGC2018-098813-B-C31, PID2022-137461NB-C32 and ERDF funds. Also, the authors would like to thank Dr. Alberto Prieto, from the Department of Computer Engineering, Automation, and Robotics of the University of Granada, Spain, for his valuable collaboration in this work.

References

1. Akbari, H., Ghofrani, S., Zakalvand, P., Tariq Sadiq, M.: Schizophrenia recognition based on the phase space dynamic of EEG signals and graphical features. Biomedical Signal Processing and Control **69**, 102917 (2021). https://doi.org/10.1016/j.bspc.2021.102917
2. Aquino-Brítez, D., Ortiz, A., Ortega, J., León, J., Formoso, M.A., Gan, J.Q., Escobar, J.J.: Optimization of deep architectures for eeg signal classification: An automl approach using evolutionary algorithms. Sensors **21**(6), 2096 (2021). https://doi.org/10.3390/s21062096
3. Asensio-Cubero, J., Gan, J.Q., Palaniappan, R.: Multiresolution analysis over simple graphs for brain computer interfaces. Journal of Neural Engineering **10**(4), 21–26 (2013). https://doi.org/10.1088/1741-2560/10/4/046014
4. Choubey, H., Pandey, A.: A combination of statistical parameters for the detection of epilepsy and EEG classification using ANN and KNN classifier. Signal, Image and Video Processing **15**(3), 475–483 (2021). https://doi.org/10.1007/s11760-020-01767-4
5. Ding, C., Peng, H.: Minimum redundancy feature selection from microarray gene expression data. In: Computer Society Bioinformatics Conference. pp. 523–528. CSB'2003, IEEE, Stanford, CA, USA (Aug 2003). https://doi.org/10.1109/CSB.2003.1227396

6. Ding, F., Wienke, S., Zhang, R.: Dynamic MPI parallel task scheduling based on a master-worker pattern in cloud computing. International Journal of Autonomous and Adaptive Communications Systems **8**(4), 424–438 (2015). https://doi.org/10.1504/IJAACS.2015.073191
7. Dong, Y., Chen, J., Yang, X., Deng, L., Zhang, X.: Energy-oriented openmp parallel loop scheduling. In: 6th International Symposium on Parallel and Distributed Processing with Applications. pp. 162–169. ISPA'2008, IEEE, Sydney, NSW, Australia (Dec 2008). https://doi.org/10.1109/ISPA.2008.68
8. Freitag, C., Berners-Lee, M., Widdicks, K., Knowles, B., Blair, G., Friday, A.: The climate impact of ict: A review of estimates, trends and regulations. arXiv (2021). https://doi.org/10.48550/ARXIV.2102.02622
9. González, J., Ortega, J., Escobar, J.J., Damas, M.: A lexicographic cooperative coevolutionary approach for feature selection. Neurocomputing **463**, 59–76 (2021). https://doi.org/10.1016/j.neucom.2021.08.003
10. Gvozdetska, N., Globa, L., Prokopets, V.: Energy-efficient backfill-based scheduling approach for SLURM resource manager. In: 15th International Conference on the Experience of Designing and Application of CAD Systems. pp. 1–5. CADSM'2019, IEEE, Polyana, Ukraine (Feb 2019). https://doi.org/10.1109/CADSM.2019.8779312
11. Ibrahim, S., Djemal, R., Alsuwailem, A.: Electroencephalography (EEG) signal processing for epilepsy and autism spectrum disorder diagnosis. Biocybernetics and Biomedical Engineering **38**(1), 16–26 (2018). https://doi.org/10.1016/j.bbe.2017.08.006
12. Jo, I., Lee, S., Oh, S.: Improved measures of redundancy and relevance for mRMR feature selection. Computers **8**(2), 42 (2019). https://doi.org/10.3390/computers8020042
13. León, J., Escobar, J.J., Ortiz, A., Ortega, J., González, J., Martín-Smith, P., Gan, J.Q., Damas, M.: Deep learning for eeg-based motor imagery classification: Accuracy-cost trade-off. PLoS ONE **15**(6), e0234178 (2020). https://doi.org/10.1371/journal.pone.0234178
14. Li, M., Xu, H., Liu, X., Lu, S.: Emotion recognition from multichannel EEG signals using K-nearest neighbor classification. Technology and Health Care **26**(S1), 509–519 (2018). https://doi.org/10.3233/THC-174836
15. Raudys, S.J., Jain, A.K.: Small sample size effects in statistical pattern recognition: Recommendations for practitioners. IEEE Transactions on Pattern Analysis and Machine Intelligence **13**(3), 252–264 (1991). https://doi.org/10.1109/34.75512
16. Sabancı, K., Koklu, M.: The classification of eye state by using kNN and MLP classification models according to the EEG signals. International Journal of Intelligent Systems and Applications in Engineering **3**(4), 127–130 (2015). https://doi.org/10.18201/ijisae.75836
17. Saeedi, M., Saeedi, A., Maghsoudi, A.: Major depressive disorder assessment via enhanced K-nearest neighbor method and EEG signals. Physical and Engineering Sciences in Medicine **43**(3), 1007–1018 (2020). https://doi.org/10.1007/s13246-020-00897-w
18. Sharma, H., Sharma, K.: An algorithm for sleep apnea detection from single-lead ECG using hermite basis functions. Computers in Biology and Medicine **77**, 116–124 (2016). https://doi.org/10.1016/j.compbiomed.2016.08.012
19. Zainuddin, A.Z.A., Mansor, W., Khuan, L.Y., Mahmoodin, Z.: Classification of EEG signal from capable dyslexic and normal children using KNN. Advanced Science Letters **24**(2), 1402–1405 (2018). https://doi.org/10.1166/asl.2018.10758