Computing
https://doi.org/10.1007/s00607-023-01193-7

REGULAR PAPER

A distributed and energy-efficient KNN for EEG classification with dynamic money-saving policy in heterogeneous clusters

Juan José Escobar[1] · Francisco Rodríguez[2] · Beatriz Prieto[2] · Dragi Kimovski[3] · Andrés Ortiz[4] · Miguel Damas[2]

Received: 4 March 2023 / Accepted: 14 June 2023

Abstract

Due to energy consumption's increasing importance in recent years, energy-time efficiency is a highly relevant objective to address in High-Performance Computing (HPC) systems, where cost significantly impacts the tasks executed. Among these tasks, classification problems are considered due to their great computational complexity, which is sometimes aggravated when processing high-dimensional datasets. In addition, implementing efficient applications for high-performance systems is not an easy task since hardware must be considered to maximize performance, especially on heterogeneous platforms with multi-core CPUs. Thus, this article proposes an efficient distributed *K*-Nearest Neighbors (KNN) for Electroencephalogram (EEG) classification that uses minimum Redundancy Maximum Relevance (mRMR) as a feature selection technique to reduce the dimensionality of the dataset. The approach implements an energy policy that can stop or resume the execution of the program based on the cost per Megawatt. Since the procedure is based on the master-worker scheme, the performance of three different workload distributions is also analyzed to identify which one is more suitable according to the experimental conditions. The proposed approach outperforms the classification results obtained by previous works that use the same dataset. It achieves a speedup of 74.53 when running on a multi-node heterogeneous cluster, consuming only 13.38% of the energy consumed by the sequential version. Moreover, the results show that financial costs can be reduced when energy policy is activated and the importance of developing efficient methods, proving that energy-aware computing is necessary for sustainable computing.

Extended author information available on the last page of the article

Published online: 27 June 2023

Keywords Parallel and distributed programming · Heterogeneous clusters · Energy-aware computing · EEG classification · KNN · Money-saving

Mathematics Subject Classification 68W15

1 Introduction

In general, society is unaware of the enormous impact of Information and Communications Technology (ICT) on greenhouse gas emissions caused by energy consumption. Its constant increase is due to the significant proliferation of electronic devices and applications frequently used for routine tasks. Also, the Internet of Things (IoT) paradigm has caused the appearance of new devices that, despite their low individual consumption, have a significant global impact given their enormous quantity. According to the predictive models on electricity use by ICT developed by Andrae and Edler [1], consumption of these technologies will increase from 13% of global electricity use in the world in 2022 to 21% in 2030 and could reach more than half (51%) of the earth's total demand in the worst-case scenario. This would mean that ICT could contribute up to 23% of global greenhouse gas emissions in 2030 [2]. Therefore, it is necessary to analyze different options, as in this work, to reduce the contribution of ICT to environmental impact. Fortunately, the most pessimistic forecasts of the ICT are not being fulfilled since consumption is lower than expected. This is because the industry around ICT is aware of the problem and is investing resources in developing policies to reduce consumption. Another way to reduce the carbon footprint produced by ICT is through the electric tariff with hourly discrimination [3]. This consists of executing the applications during the hours when the tariff is lower, and the wind and/or solar electricity production is higher. The result is that the economic cost of the energy necessary for the execution of the programs could decrease, and the use of alternative fuels is also encouraged. Some countries digitally report the energy price in real-time, making it possible to combine this information with scheduled executions of the applications. This is one of the aspects dealt with in this work. Specifically, the contributions of this paper are the following:

- Provide an energy-efficient, parallel, and distributed exploration approach based on mRMR+KNN that exploits heterogeneous clusters with Non-Uniform Memory Access (NUMA) nodes.
- Apply mRMR for feature selection and KNN for subsequent classification to solve an EEG classification problem that involves a dataset of the University of Essex. The mRMR+KNN combination had not previously been applied to that dataset.
- Analyze the proposed application from three fundamental metrics: energy consumption, execution time, and accuracy of the results. The study aims to show how greater energy efficiency can be achieved by adequately exploiting the hardware architecture on which the application runs. This is especially useful to save time and energy in future research based on the mRMR+KNN combination.

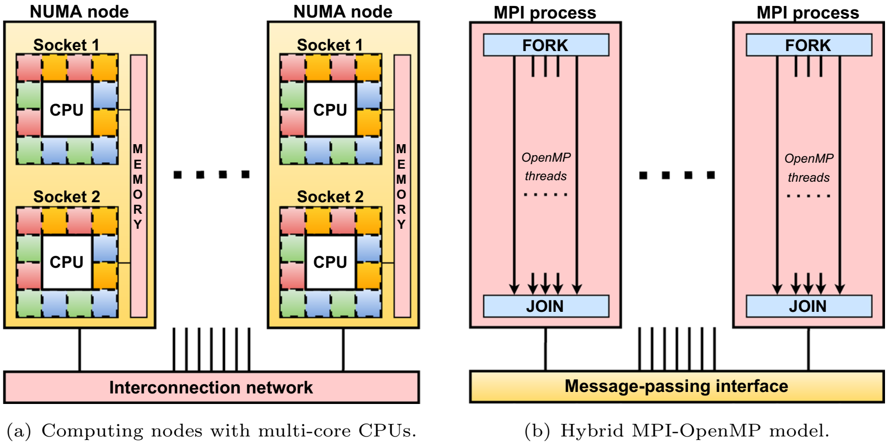
- Provide an energy policy to save money or energy, depending on the user preferences. The policy is intended to be easily adaptable to other HPC applications.

The approach developed in this work implements a hybrid MPI-OpenMP model to achieve higher performance by increasing parallelism and minimizing communications and overhead. Message-Passing Interface (MPI) is mainly used for inter-node communications and OpenMP for multi-threaded parallelization in each node. Figure 1 shows how the implementation of the proposed approach (Fig. 1b) maps to the physical architecture of the clusters for which the application has been designed (Fig. 1a). In this sense, each MPI process is in charge of (i) managing the execution of a NUMA node; (ii) exchange information between nodes through the network, and (iii) process their part of the work by distributing computational load between the different CPU cores with OpenMP threads.

After this introduction, the rest of the document is structured as follows: Sect. 2 reviews different works in the literature related to the topic addressed in this paper. Section 3 details the different parallel implementations of the proposed approach and its energy policy for money-saving. Then, Sect. 4 analyzes the experimental results and discusses the importance of energy awareness in HPC systems. Finally, Sect. 5 provides the paper's conclusions.

2 Related work and background

Given the great importance that a good balance between performance and energy efficiency has in computing, different methods and solutions have been proposed to address this problem. The work [4] compiles an extensive list of works on HPC and categorizes them based on compute device type, optimization metrics, and energy



(a) Computing nodes with multi-core CPUs.

(b) Hybrid MPI-OpenMP model.

Fig. 1 Architecture of HPC clusters and the corresponding hybrid MPI-OpenMP model implemented in this work to take advantage of their computing nodes

methods. The compatibility of employing techniques for HPC systems, such as workload balancing or Dynamic Voltage and Frequency Scaling (DVFS), has also been discussed in several studies [5, 6]. The environmental impact produced by ICT is also being combated in different ways. Among those actions are [7]:

- *Technological improvements in electronic components* The increase in the number of devices in the ICT makes the energy efficiency of the devices acquire greater relevance. Changing the internal architecture of microchips has been some outstanding action in this field. For example, special-purpose processors such as Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs) have been developed, which have turned out to be very efficient in specific applications. Neuromorphic computing [8] and the integration of cooling directly into the chip using microfluidic systems [9] are also booming. The change from Hard Disk Drive (HDD) to Solid State Drive (SSD) technology has significantly reduced energy consumption in mass data storage.
- *Scheduling and resource management* The objective is to use the different resources available in the system to reduce energy consumption. For example, implementing power management features to dynamically switch between different power states [10], depending on the workload, is a suitable option to make data centers more energy efficient [11]. Power could be saved in different ways: by using standby modes on resources that are not currently needed [12], or by setting a hardware energy consumption cap [13]. The use of parallelism makes it possible to speed up applications and reduce their energy consumption by avoiding idle processing cores.
- *Scale changes* This is intended to migrate small specialized systems (calculators, alarm clocks, etc.) to more energy-efficient equipment like smartphones. In distributed systems, the energy consumption of consumer devices is decreasing because application execution is offloaded to networks and data centers [14].

Regarding bioinformatics, this field has experienced exponential growth in recent years. As a result, biological datasets have grown considerably in size, as in the case of Electroencephalography. This discipline deals with EEG signals, which represent the electrical activity of different brain parts. For example, EEG signals are used to aid in the diagnosis of disorders such as schizophrenia [15], dyslexia [16], depression [17], autism [18], sleep problems [19], epilepsy [20] and seizure manifestations, in general [21–24]. They are also used to classify motor functions, such as movement of limbs or eyes [25], and for the classification of human emotions [26]. Some emerging areas of Artificial Intelligence, such as Machine Learning, aim to recognize patterns in these signals for their subsequent classification [27]. This tasks can be addressed through various Machine Learning methods using non-brain-inspired or brain-inspired techniques. KNN algorithm, used in this work, belongs to the first ones. In this context, this paper considers a case that falls within the scope of Motor-Imagery (MI) classification due to its social interest in Brain-Computer Interface (BCI) tasks [28]. The problem consists of identifying from EEG signals different motor-imagery movements without its real execution [29]. However, the main problem of working with EEG signals is their high dimensionality, which makes their correct prediction difficult since most of them do not contain relevant information. Therefore, it is important to apply feature selection techniques, such as mRMR, to obtain the most relevant ones.

Among the different existing datasets in the literature, this work focuses on a dataset [30] that belongs to the BCI laboratory of the University of Essex. It correspond to a human subject coded as 104 and includes 178 EEG 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 proposed approach deals with a 3-class classification problem.

There are different works that have dealt with this dataset. The first one [30] analyzed the performance of the Linear Discriminant Analysis (LDA) classifier depending on the MultiResolution Analysis (MRA) approach used to preprocess the data. In addition, they propose a new MRA method called Graph Lifting Scheme (GLS), which is compared with others such as Linear Lifting Scheme (LLS), Db5, and Haar. Subsequently, in the works [31, 32], three alternatives were analyzed: OPT1, OPT2, and OPT3, which carry out the classification process by varying the number of LDA classifiers according to the majority voting. In [33, 34], two filter methods that use Non-dominated Sorting Genetic Algorithm II (NSGA-II) for multi-objective feature selection are proposed. In this case, the filter method is based on a set of label-aided utility functions that do not require the accuracy or the generalization of the classifier. The procedure defines a function for each label in the classification problem that is used as the objective (or fitness) function by NSGA-II. The paper [35] analyzes the dataset using Sparse Representation (SR) in combination with LDA and Support Vector Classifier (SVC), while [36] presents LeOCCEA, a wrapper procedure that hybridizes concepts of Cooperative Co-Evolutionary Algorithms (CCEAs) and lexicographic optimization to make possible the simultaneous optimization of two interdependent problems: finding the best hyperparameter values for the classifier applied within the wrapper method while minimizing the number of features that best describe the dataset. The wrapper is compared with other classifiers such as Support Vector Machine (SVM), Naive Bayesian Classifier (NBC), and KNN. It is also used as a feature selection method prior to using these classifiers.

The application of more modern classification methods to these datasets, such as neural networks, has been studied in [37–39], where [38, 39] also provides measures of energy consumption and execution time. On the one hand, [37] analyzes the performance of a Deep Belief Network (DBN) when combined with LDA to reduce the dimensionality of the datasets. On the other hand, in [38] several Convolutional Neural Networks (CNNs), Feed-Forward Neural Networks (FFNNs), and Recurrent Neural Networks (RNNs) are analyzed when their hyperparameters are optimized (or not) by means of a Genetic Algorithm (GA). It is also studied whether previously applying feature selection to the dataset through logistic regression affects the quality of the results. Finally, in [39] a framework is proposed to automatically optimize the hyperparameters of the classifiers through an NSGA-II. The results exceed those obtained by the EEGNet [40] and DeepConvNet [41] neural networks without

Table 1 Kappa values from previous works dealing with the University of Essex dataset 104

Works	Approach	Average ± standard deviation
Asensio-Cubero et al. [30]	Haar	0.421
	Db5	0.361
	LLS	0.453
	GLS	**0.790**
Ortega et al. [31]	OPT1	0.698 ± 0.062
Ortega et al. [32]	OPT2	0.614
	OPT3	0.606
Martín-Smith et al. [33, 34]	FOPT1	0.639 ± 0.045
	FOPT1*	0.661 ± 0.015
Ortega et al. [35]	SR+LDA	0.696
	SR+SVC	0.664
González et al. [36]	KNN	0.704 ± 0.031
	NBC	0.642 ± 0.029
	LDA+KNN	0.647 ± 0.053
	LDA+NBC	0.677 ± 0.047
	LeOCCEA+SVM	0.578 ± 0.046
	LeOCCEA+KNN	0.543 ± 0.053
	LeOCCEA+NBC	0.639 ± 0.061
Ortega et al. [37]	DBN	0.733
	DBN+opt	0.750
	LDA+DBN	0.651 ± 0.052
León et al. [38]	CNN	0.729
	CNN+opt	0.754
	CNN+fs	0.635
	FFNN	0.587
	FFNN+fs	0.678
	FFNN+opt+fs	0.728
	RNN	0.610
	RNN+fs	0.677
	RNN+opt+fs	0.712
Aquino-Brítez et al. [39]	DeepConvNet	0.38 ± 0.04
	EEGNet	0.44 ± 0.08
	EEGNet+opt	0.63 ± 0.01

The best value is shaded bold

fs Feature selection; *opt* Hyperparameter optimization

optimization. A complete comparison of the results of all the methods mentioned above can be seen in Table 1.

As can be seen, none of the previous works applied the mRMR+KNN combination to the University of Essex dataset, although the KNN algorithm has been studied in [36]. KNN classifier has been chosen in this work due to its good effectiveness in motor-imagery applications, as shown in [42]. However, KNN is very sensitive to the curse of dimensionality, worsening its performance with high-dimensional datasets. Thus, as mRMR is able to reduce the dimensionality of the patterns and KNN has proven to be efficient with a small number of input features [36], the combination of mRMR+KNN could achieve good results. In addition, mRMR has shown great effectiveness in other fields of biomedicine, such as those dealing with microarray gene expression data [43].

3 The proposed approach

EEG classification has been approached using the KNN algorithm together with the mRMR technique [44]. KNN 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 training ones. 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.
4. Assign the new instance to the predominant class.

Although interpretability is the main advantage of this method, the computing time depends linearly on the dataset size since each new instance must be compared with all the training ones. In this regard, multiple improvements over the basic procedure have been proposed [45]. However, one of the most useful ways to reduce execution times without decreasing accuracy is to use parallelism techniques. Parallelism is especially useful in clusters that include multiprocessors and accelerators, such as GPUs, since it allows us to take advantage of these devices' potential. Regarding mRMR, this method tends to select the subset of features that has the highest correlation to the class (output) and the lowest correlation between them. It ranks the N_F features of the datasets according to the minimum-redundancy-maximum-relevance criterion, whose implementation in this work is based on the one proposed in [43] and takes the F-test Correlation Quotient (FCQ) to select the next feature. mRMR helps to deal with the dimensionality problem [46] and to reduce computation time by avoiding the evaluation of all 2^{N_F} possible subsets of features. Instead, the proposed approach only evaluates N_F of them, where in each one, the next feature from the list provided by mRMR is added. For each subset, all possible values of the K parameter are evaluated to get the best classification accuracy.

Algorithm 1: 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      $N_I \leftarrow$ getNumberInstances($Te$)
       // Vector with the correct predictions for each value of $K$
3      $C_P \leftarrow \{0\}$
       // Set as many OpenMP threads as logical CPU cores
4      #pragma omp parallel for
5      for $i \leftarrow 1$ to $N_I$ test rows do
           // Distance between instance $te[i]$ and the training ones
6          $D \leftarrow$ calculateDistances($Te[i], Tr, Idx$)
7          for $k \leftarrow 1$ to $N_I$ do
8              $P_C \leftarrow$ predominantClass($k, D$)
9              if prediction $P_C$ is correct then
10                 $C_P[k]{+}{+}$
11             end
12         end
13     end
14     $C_P \leftarrow$ sort($C_P$, "Descending")
15     $acc \leftarrow \frac{C_P[1]}{N_I}$
16     return $acc$
17 End
```

Since finding the best K and feature subset is computationally complex, the algorithm has been parallelized to reduce execution time and, in turn, energy consumption. In fact, parallelization occurs at two levels through a hybrid approach with MPI and OpenMP libraries: distributing subsets among worker nodes with MPI 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).

The algorithmic complexity of a standard KNN to classify all instances of the test dataset is $\mathcal{O}(K \cdot Tr \cdot Te \cdot N_F)$, where K is the number of neighbors of the KNN algorithm, Tr the number of instances in the training dataset, Te the number of instances in the test dataset, and N_F the data dimensionality. However, the proposed approach evaluates all the K possible values for each subset of features, which have a different computational load since the number of features in each subset is greater. As the computational load also grows when increasing the value of K (more neighbors to compare), the complexity of the procedure when executed on a single-processor machine is $\mathcal{O}(K \cdot \log_2(K) \cdot Tr \cdot Te \cdot N_F \cdot \log_2(N_F))$. Taking into account that the program takes advantage of all processors and is designed to be executed on many multi-core machines, the final time complexity of the proposed approach is $\frac{\mathcal{O}(K \cdot \log_2(K) \cdot Tr \cdot Te \cdot N_F \cdot \log_2(N_F))}{P \cdot N_{Wk}}$, being P the number of processors (CPU threads) and N_{Wk} the number of worker nodes.

3.1 A distributed master-worker scheme for node-level parallelization

Algorithm 2: Master-worker approach and the energy policy.

```
Function Main($Tr, Te, C_S, N_{Wk}$)
    Input  : Training and test datasets, $Tr$ and $Te$
    Input  : Max 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, $bestAcc$
    $bestAcc \leftarrow -1$
    if $Master$ then
        $N_F \leftarrow$ getNumberFeatures($Tr$)
        $indexList \leftarrow \{1, \ldots, N_F\}$
        $stops \leftarrow 0$
        while $stops \neq N_{Wk}$ do
            MPI::Recv($acc, tag$)
            if $tag$ is RESULT and $acc > bestAcc$ then
                $bestAcc \leftarrow acc$
            end
            // Next chunk to send, e.g. indices 7,8,9 when $C_S = 3$
            $chunk \leftarrow$ getNextFeatureChunk($indexList, C_S$)
            if size($chunk$) $> 0$ then
                MPI::Send($chunk$, JOB_DATA)
            else
                MPI::Send(NULL, STOP)
                $stops \leftarrow stops + 1$
            end
        end
    else
        $rankedFeatures \leftarrow$ mRMR($Tr$)
        $Tr, Te \leftarrow$ sortDatasets($Tr, Te, rankedFeatures$)
        MPI::Send(NULL, FIRST_JOB)
        MPI::Recv($chunk, tag$)
        while $tag$ is JOB_DATA do
            for $i \leftarrow 1$ to size($chunk$) do
                $price \leftarrow$ checkEnergyAPI()
                while $price$ is not cheap do
                    sleepUntil(timeToNextHour())
                    $price \leftarrow$ checkEnergyAPI()
                end
                $acc \leftarrow$ evaluateFeatureSubset($Tr, Te, chunk[i]$)
                if $acc > bestAcc$ then
                    $bestAcc \leftarrow acc$
                end
            end
            MPI::Send($bestAcc$, RESULT)
            MPI::Recv($chunk, tag$)
        end
    end
    return $bestAcc$
End
```

Table 2 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

The proposed application, 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 MPI process #0 is assigned to the master process and the rest to the workers. The algorithm receives the input parameters: the datasets, the number of workers, and the maximum chunk size to send to the workers. The execution ends when there is no more work to process and the function returns the best accuracy found (Line 41). 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 2. 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). As all workers must receive the tag to finish, the master must track how many workers have received it (Line 17).

Regarding the workers (Lines 20 to 40), they apply the mRMR algorithm on the training dataset to obtain the ranked list of features. As the index of features changes, datasets are reordered to speed up computation by making use of *Coalescing* [47] (Line 22). This technique allows multiple memory accesses to be combined into a single transaction. At this point, a worker is ready to request jobs by sending a message with the `FIRST_JOB` tag (Line 23). For each chunk of features received (Lines 26 to 36), if the energy price is cheap the worker obtains the accuracy of the corresponding feature subset by calling in Line 32 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 37 and 38). This process is repeated until the `STOP` tag is received, indicating that the worker can end its execution.

3.2 Implementation of the energy policy for money-saving

The proposed master-worker algorithm also incorporates an energy policy that automatically pauses the algorithm during the hours when the energy price is higher and resumes execution when the price is lower. Consequently, this policy would allow data centers to save money but at the cost of extending the execution time and the energy consumed, although the computing devices would still be on even if the algorithm is paused. This policy is a client that opens a secure communication socket with the energy Application Programming Interface (API) [48]. In this way, the client sends HyperText Transfer Protocol (HTTP) requests to port 443 to obtain the price of energy according to the regulated tariff of the Spanish energy market, called Voluntary Price for Small Consumers (VPSC). The API returns a data structure that contains attributes associated with the energy market. Among them, those used by the policy are:

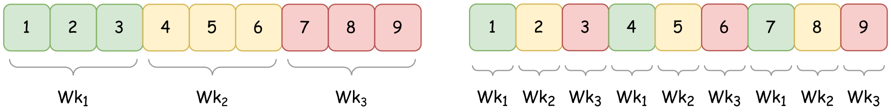
- The price of energy, expressed in €/MW·h.
- A parameter indicating whether the price of energy is low.
- A parameter indicates whether the energy price is below the average for the day.

The user can disable the policy by unchecking the money-saving option in the program's configuration file. Otherwise, the MPI process that controls each worker node will be in charge of checking the price of energy in two possible circumstances (see Algorithm 2):

- When the worker receives a work chunk from the master. If the energy price at that moment is expensive, the execution is paused, and the MPI process sleeps until the next hour. Through the program's configuration file, the user can indicate that values below the daily average are also considered cheap or define a custom price threshold. However, this is risky since if the specified value is not in the range of upcoming prices, the algorithm will never start.
- When the MPI process is asleep and the next hour is reached. In this case, the process wakes up, checks the price, and resumes execution if the price is acceptable or goes back to sleep until the next hour (when the tariff changes).

3.3 Ways of distributing the workload

As previously seen in Sect. 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 [49]. 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 blocks (see Fig. 2).


1
2
3
4
5
6
7
8
9
Wk$_1$
Wk$_2$
Wk$_3$
1
2
3
4
5
6
7
8
9
Wk$_1$ Wk$_2$ Wk$_3$ Wk$_1$ Wk$_2$ Wk$_3$ Wk$_1$ Wk$_2$ Wk$_3$

(a) Contiguous blocks (b) Strided blocks

Fig. 2 The two different static workload distributions used by the master node

The strided assignment could reduce the workload imbalance [50] present in the contiguous blocks alternative since each node would compute similar subsets. The impact of the different workload distributions and the chunk size on performance is discussed in Sect. 4.

4 Experimental work

This section analyzes the classification results obtained by the proposed approach and compares the energy-time performance of each workload distribution when using multiple nodes. In addition, a benchmark is performed to demonstrate the benefits of using the energy policy when computing over long periods. All experiments are repeated ten times to obtain more reliable measurements of the application's behavior.

4.1 Experimental setup

The application has been executed in an HPC cluster composed of eight heterogeneous Non-Uniform Memory Access (NUMA) nodes interconnected via Gigabit Ethernet and whose CPU devices are detailed in Table 3. The cluster runs the *Rocky Linux* distribution (v8.5) and schedules the jobs using the *Slurm* task manager (v20.11.7) [51]. The *C*++ source codes have been compiled with the GCC compiler (v8.5.0), the OpenMPI library (v4.0.5) with support for the MPI API (v3.1.0), and optimization flags `-O2 -funroll-loops`.

The energy measurements of each node have been obtained from a custom wattmeter called *Vampire*, which is based on the ESP32 microcontroller [52], capable of capturing in real-time information of instantaneous power (W) and accumulated energy consumed (W · h). This meter allows the monitoring of multiple computers in a synchronized and independent way, accurately measuring the energy consumption of a program that is running in a distributed manner in a multi-node cluster. Since the meter reads the voltage (V) and current (A) every second, the total energy consumed by the application over a given time can be determined. Although the data obtained is transmitted to a remote server via WiFi or Bluetooth and stored in an InfluxDB database for further processing, it can be viewed in real-time using the Grafana user interface.

Table 3 Characteristics of the cluster used in the experiments

Node	CPU				RAM	
	Model	Total cores/ threads	TDP[1] (W)	Frequency (MHz)	Frequency (MHz)	Size (GB)
Master	1x Intel Xeon E5-2620 v4	12/24	160	2100	1600	32
1	1x Intel Xeon E5-2620 v4	8/16	85		2133	
2	2x Intel Xeon E5-2620 v4	16/32	170			
3 to 7	2x Intel Xeon Silver 4214	24/48	170	2200	2933	64

[1] The Thermal Design Power (TDP) parameter indicates the amount of heat that a chip will generate in normal operation, measured in watts (W), which is often used to estimate the energy consumption of the chip

Table 4 Comparison of Kappa values between the proposed approach and the top 3 of Table 1

Works	Approach	Value
This work	mRMR+KNN	**0.83**
Asensio-Cubero et al. [30]	GLS	0.790
León et al. [38]	CNN+opt	0.754
Ortega et al. [37]	DBN+opt	0.750

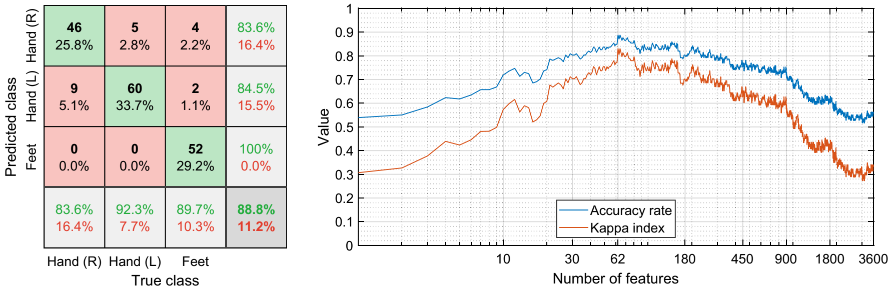
The best value is shaded bold

opt Hyperparameter optimization.

4.2 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 (see Table 4). 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 3a 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 Fig. 3b. 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 irrelevant. It is also observed that the values of accuracy and Kappa distance themselves for extreme values of the graph.

The statistical validation of the results has been carried out using 10,000 iterations of a permutation test. This ensures that the results obtained are not achieved by



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

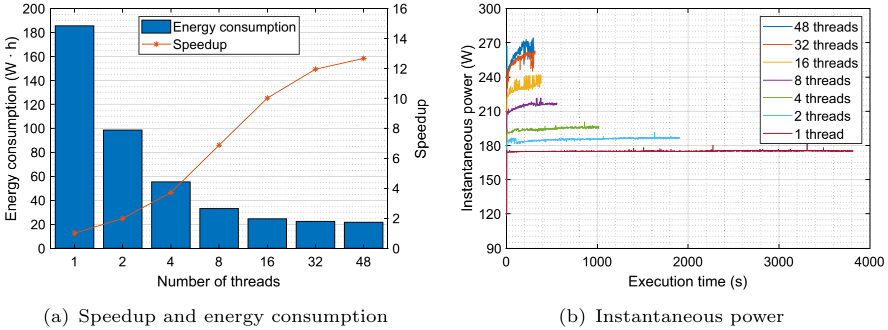
(b) Accuracy rate and Kappa index when adding features

Fig. 3 Classification results of the proposed approach when using mRMR and $K = 18$

chance when testing the null hypothesis. In our case, the null hypothesis consists in obtaining higher performance values (Kappa index values) in the permuted dataset than in the non-permuted one. In the test, a p-value $< 1.10^{-7}$ has been obtained, showing that the null hypothesis can be rejected since its value is less than 0.01. Despite the good results, there are limitations inherent to the use of the KNN classifier in the global optimization method proposed in this work. For example, it is well-known that the appropriate value of K can vary greatly depending on the dataset used, especially if it is characterized by the curse of dimensionality, which can deteriorate the generalizability of the KNN classifier when few samples are available. However, this is a common limitation for most classifiers and should not be a particular drawback compared to other classical alternatives, although classifiers such as SVM have proven to be efficient with high-dimensional datasets. An example of this can be seen in [53], where an adaptive SVM is used in conjunction with neural networks to address the BCI Competition IV 2a dataset. It could be interesting to study if this approach could be applied to the dataset addressed in this work since both datasets share similarities, although it is also worth noting that KNN is less prone to overfitting than classifiers based on neural networks. There are other types of limitations as well. On the one hand, deal with unbalanced datasets: KNN assumes that instances within the same class are grouped together. However, in some cases, the decision boundaries may be complex or non-linear, and KNN may have difficulty capturing them accurately. In other words, unbalanced data can lead to biased predictions. In the dataset discussed here, the classes are balanced, without any appreciable bias (33%, 37%, and 30%). On the other hand, for very large datasets, the computational load and memory consumption associated with the training process could be a limitation, since it requires the comparison of a test instance with all the training samples.

4.3 Energy-time performance

Figure 4 shows the application's performance after running on Node 3 without using the power policy. 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



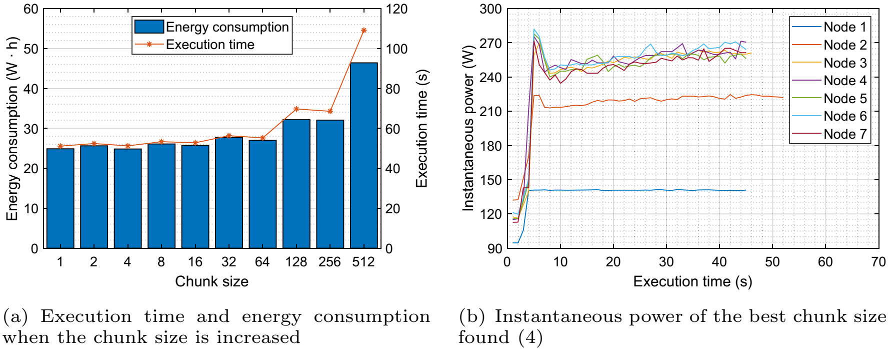
(a) Speedup and energy consumption

(b) Instantaneous power

Fig. 4 Performance obtained by the proposed approach in a single-node configuration when varying the number of OpenMP threads

cores is increased. From Fig. 4a, 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 in [54], 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 Fig. 4b). However, energy consumption also depends linearly on execution time, and since speedup increases at a greater rate than energy, total energy consumption is less. This behavior has been widely demonstrated in the literature for a wide variety of parallel and distributed applications.

The performance of the hybrid MPI-OpenMP approach that corresponds to the second level of parallelism is shown in Figs. 5 and 6. Again, the power policy has not been activated in order to measure the performance scalability of the application. On the one hand, Fig. 5 exposes the behavior of the application when all nodes are used, and the workload distribution is dynamic. Figure 5a reveals that a very large chunk size leads to worse execution time and energy consumption mainly due to workload imbalance. It is also noteworthy that a chunk size of 1 provides good results, which suggests that the cost of communications in this application is very low. The instantaneous power of each node for a chunk size of 4 is plotted in Fig. 5b. Although 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



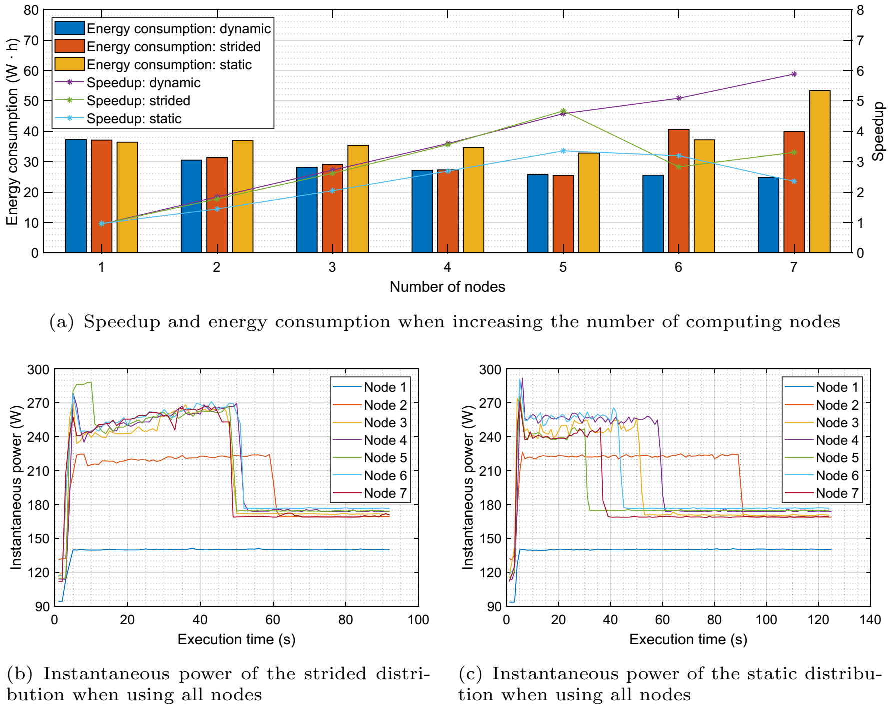
(a) Execution time and energy consumption when the chunk size is increased

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

Fig. 5 Performance obtained by the dynamic workload distribution when using all nodes

observed in the figure, and in the other 9 repetitions of this experiment, is that most nodes end up simultaneously, which is expected in dynamic workload distributions.

On the other hand, Fig. 6 compares the different workload distributions. The number of computation nodes indicated in Fig. 6a does not correspond to the order shown in Table 3. Instead, the nodes in the graph correspond to homogeneous and



(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. 6 Comparison of performance between the different workload distributions

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 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 Fig. 6c 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 (Fig. 6b), 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. Extrapolating the data, if all the nodes were homogeneous, a speedup of approximately 6.4 could be achieved.

It should also be noted that the speedup has been calculated with respect to the time depicted in Fig. 4b, where the master-worker scheme does not exist and therefore Node 3 does all the work. The objective is that the data shown in the figure take into account the overhead caused by the existence of the master and its communications with the worker nodes involved. As a consequence, all speedup values are somewhat less than those calculated based on the single-node time of the dynamic distribution. For this reason, and although it is difficult to see in the figure, the speedup of the 3 distributions is slightly below 1 when one computing node is used. Furthermore, it is the only case in which the static and strided distributions are slightly better than the dynamic one. This makes sense because if only one node computes, there is no need to continuously distribute chunks of work.

4.4 Energy policy benchmark

As discussed in Sect. 3.2, the energy policy checks the price of energy every hour and decides whether to pause, continue, or resume the execution of the application. However, its execution time, as has been observed in previous figures, sometimes does not even reach one hour. In order to measure the energy and economic impact of the policy, the application has been running in a loop for a total of 24 h using dynamic distribution and 7 nodes, each computing with the maximum number of OpenMP threads available. Figure 7 shows the price of energy and the sum of the instantaneous power of all the nodes during the execution. In the slots where the price is cheaper (shown in green), the instantaneous power increases, meaning that

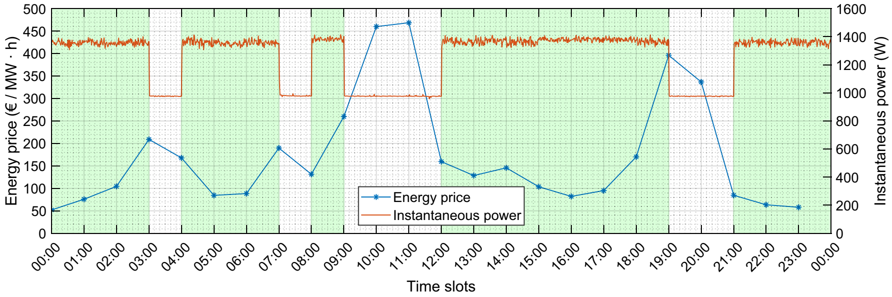


Fig. 7 Activity of the nodes and energy price for a 24-hour period. The cheap time slots, colored green, are those where the price is below the daily average (171.48 €/MW · h)

the nodes are computing. This can be seen in the slots [00:00–03:00, 04:00–07:00, 08:00–09:00, 12:00–19:00, 21:00–00:00]. In the rest of them, the nodes pause their execution, and therefore the power drops. Now suppose the need to compute a task that requires 6 h of execution without the use of the policy. Depending on the time window in which it runs, using the policy could save or lose money for the user as the total number of hours needed could be much higher. This is analyzed in Table 5.

The results obtained are very different. On the one hand, for the first seven time slots, using the policy loses money. In the first two, despite the fact that the computation takes an hour or two more, the price of energy in the expensive slots is

Table 5 Energy consumption and money-saving for a 6-hour run when the policy is activated

Time slots (#)	Energy consumption (W · h)	Total cost (€)	Money-saving (%)
00:00–07:00 (7)	8,917.62	0.094	– 0.35
01:00–09:00 (8)	9,827.05	0.122	– 23.42
02:00–13:00 (11)	12,364.64	0.236	−106.29
03:00–14:00 (11)	12,377.91	0.239	– 102.27
04:00–14:00 (10)	11,501.40	0.221	−76.66
05:00–15:00 (10)	11,500.09	0.218	– 32.56
06:00–16:00 (10)	11,527.15	0.221	– 2.00
07:00–17:00 (10)	11,551.10	0.220	**2.81**
08:00–18:00 (10)	12,049.94	0.217	**0.84**
09:00–18:00 (9)	10,666.98	0.199	**9.92**
10:00–18:00 (8)	9,790.01	0.176	**11.89**
11:00–18:00 (7)	8,913.34	0.136	**8.77**
12:00–18:00 (6)	8,037.21	0.095	0
13:00–19:00 (6)	8,038.20	0.096	0
14:00–22:00 (8)	9,777.85	0.154	– 13.84
15:00–23:00 (8)	9,875.08	0.145	**10.29**
16:00–24:00 (8)	9,854.34	0.138	**12.78**

The cases where financial costs are reduced are shaded bold only slightly higher than in the cheap ones, making it not worth it. The same, but in a more extreme form, occurs in the following four. What happens here is that in all execution windows, in addition to the above, there is also a period of three consecutive expensive slots, causing the amount of energy consumed to be much more than if the policy were not used. On the other hand, for the rest of the time slots in the table, the policy saves money or at least does not lose it, except for the slot [14:00–22:00]. The reason why in the slots [12:00–18:00, 13:00–19:00] the saving is 0 is that the number of computing hours, whether the policy is used or not, is the same (6 h). Finally, it must be taken into account that the results have been obtained considering cheap areas whose price is below the daily average. However, as discussed in Sect. 3.2, the application allows the user to configure the threshold that defines each slot. This means that the results could be worse or better depending on the threshold value, so a study that determines the optimal threshold is essential. Also, the user could estimate the best time of day to run the program: the energy price is known a priori and the energy consumption both in idle and active states always has the same pattern.

5 Conclusions and future work

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, contrary to other works that focus only on the accuracy of the results and on the execution time. Experiments have been carried out with the synchronization of the executions in time slots with different energy costs. 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, 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 [55].

Regarding the energy policy, it allows for saving money or energy depending on the needs of the user, stopping or resuming the program execution according to the cost per megawatt. One of the advantages of this method is that it does not depend on the implemented application, so it can be easily adapted to other applications without major changes. Another advantage is that running the algorithm in the lowest cost time slots also contributes to reducing greenhouse gas emissions, since during these periods the generation of energy from renewable sources (wind, photovoltaic, hydraulic, etc.) is greater. However, while it has proven useful under some circumstances, the following drawbacks have been identified:

- The price difference between expensive or cheap slots must be substantial to offset the energy consumption of overtime.

- A period of consecutive expensive slots makes it difficult to save money since the total computing time is much higher.
- It is not possible to save if the proportion of expensive slots is much higher than that of cheap ones. According to other benchmarks carried out, savings can be made if the number of cheap slots is greater than 65% unless the difference in cost between the expensive and cheap periods is very abrupt. However, although it is common for cheap slots to predominate, finding abrupt differences with expensive slots is less likely. A quick solution could be to adjust the threshold to define which slots are considered expensive or cheap.

The common problem with those points is that the application consumes energy during the expensive slots without advancing the computation. In future work, different alternatives are proposed to improve the policy. One of them would consist of applying the DVFS technique to reduce the frequency and voltage of the CPU, minimizing energy consumption and, therefore, the monetary cost. The policy should consider the consumption of standby devices as one more parameter and completely turn off the devices when possible. Another improvement would be to not pause the execution of the procedure or to do it as little as possible. For example, if the system had a big.LITTLE architecture, only the so-called low-performance cores should be used during the expensive slots. This would allow computing but with moderate energy consumption, shortening the execution time and minimizing the probability of falling into expensive slots. Also, in cases where a company has processing centers in different countries, it would also be possible to dynamically reallocate the workload in those where the cost of energy is lower [56]. In this way, the execution of the procedure would never be paused, but the counterpart is that the application must be redesigned to be able to transfer its execution status and data to other computers, which entails an extra cost for communications.

Acknowledgements 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.

Author Contributions Conceptualization: JJE and FR; Methodology: JJE and MD; Software: FR; Validation: BP and DK; Formal analysis: JJE; Investigation: FR; Resources: FR and MD; Data curation: DK and AO; Writing—original draft preparation: JJE and FR; Writing—review and editing: BP and DK; Visualization: JJE and FR; Supervision: AO and MD; Project administration: AO and MD; Funding acquisition: AO and MD.

Funding Funding for open access publishing: Universidad de Granada/CBUA. This research was funded by the Spanish Ministry of Science, Innovation, and Universities under Grants PGC2018-098813-B-C31, PID2022-137461NB-C32, and the ERDF fund. Funding for open access charge: University of Granada/CBUA.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Andrae ASG, Edler T (2015) On global electricity usage of communication technology: Trends to 2030. Challenges 6(1):117–157
2. Freitag C, Berners-Lee M, Widdicks K, Knowles B, Blair G, Friday A (2021) The climate impact of ict: a review of estimates, trends and regulations. arXiv
3. Tushar W, Yuen C, Smith DB, Poor HV (2017) Price discrimination for energy trading in smart grid: a game theoretic approach. IEEE Trans Smart Grid 8(4):1790–1801
4. Czarnul P, Proficz J, Krzywaniak A (2019) Energy-aware high-performance computing: Survey of state-of-the-art tools, techniques, and environments. Sci Progr. https://doi.org/10.1155/2019/8348791
5. Wang Z, Wang H, Zhao W, Cheng L (2019) Energy optimization of parallel programs in a heterogeneous system by combining processor core-shutdown and dynamic voltage scaling. Futur Gener Comput Syst 92:198–209
6. Li H, Wei Y, Xiong Y, Ma E, Tian W (2021) A frequency-aware and energy-saving strategy based on DVFS for spark. J Supercomput 77(10):11575–11596
7. Manganelli M, Soldati A, Martirano L, Ramakrishna S (2021) Strategies for improving the sustainability of data centers via energy mix, energy conservation, and circular energy. Sustainability 13(11):6114
8. Marković D, Mizrahi A, Querlioz D, Grollier J (2020) Physics for neuromorphic computing. Nat Rev Phys 2(9):499–510
9. Wei T (2020) All-in-one design integrates microfluidic cooling into electronic chips. Nature 585:188–189
10. Feller E, Rohr C, Margery D, Morin C (2012) Energy management in iaas clouds: a holistic approach. In: 5th international conference on cloud computing. pp 204–212. CLOUD'2012, IEEE, Honolulu, HI, USA, Jun 2012
11. Hotta Y, Sato M, Kimura H, Matsuoka S, Boku T, Takahashi D (2006) Profile-based optimization of power performance by using dynamic voltage scaling on a PC cluster. In: 20th international parallel and distributed processing symposium. pp 1–8. IPDPS'2006, IEEE, Rhodes, Greece, Apr 2006
12. Lin M, Wierman A, Andrew LLH, Thereska E (2011) Dynamic right-sizing for power-proportional data centers. In: 30th annual joint conference: INFOCOM. pp 1098–1106. INFOCOM'2011, IEEE, Shanghai, China, Apr 2011
13. Lefurgy C, Wang X, Ware M (2008) Power capping: a prelude to power shifting. Clust Comput 11(1):183–195
14. Kumar K, Lu YH (2010) Cloud computing for mobile users: Can offloading computation save energy? Computer 43(4):51–56
15. Akbari H, Ghofrani S, Zakalvand P, Tariq Sadiq M (2021) Schizophrenia recognition based on the phase space dynamic of EEG signals and graphical features. Biomed Signal Process Control 69:102917
16. Zainuddin AZA, Mansor W, Khuan LY, Mahmoodin Z (2018) Classification of EEG signal from capable dyslexic and normal children using KNN. Adv Sci Lett 24(2):1402–1405
17. Saeedi M, Saeedi A, Maghsoudi A (2020) Major depressive disorder assessment via enhanced K-nearest neighbor method and EEG signals. Phys Eng Sci Med 43(3):1007–1018
18. Ibrahim S, Djemal R, Alsuwailem A (2018) Electroencephalography (EEG) signal processing for epilepsy and autism spectrum disorder diagnosis. Biocybern Biomed Eng 38(1):16–26
19. Sharma H, Sharma K (2016) An algorithm for sleep apnea detection from single-lead ECG using hermite basis functions. Comput Biol Med 77:116–124
20. Choubey H, Pandey A (2021) A combination of statistical parameters for the detection of epilepsy and EEG classification using ANN and KNN classifier. SIViP 15(3):475–483

21. Subasi A, Erçelebi E (2005) Classification of EEG signals using neural network and logistic regression. Comput Method Programs Biomed 78(2):87–99
22. Subasi A (2007) EEG signal classification using wavelet feature extraction and a mixture of expert model. Expert Syst Appl 32(4):1084–1093
23. Richhariya B, Tanveer M (2018) EEG signal classification using universum support vector machine. Expert Syst Appl 106:169–182
24. Truong ND, Nguyen AD, Kuhlmann L, Bonyadi MR, Yang J, Ippolito S, Kavehei O (2018) Convolutional neural networks for seizure prediction using intracranial and scalp electroencephalogram. Neural Netw 105:104–111
25. Sabancı K, Koklu M (2015) The classification of eye state by using kNN and MLP classification models according to the EEG signals. Int J Intell Syst Appl Eng 3(4):127–130
26. Li M, Xu H, Liu X, Lu S (2018) Emotion recognition from multichannel EEG signals using K-nearest neighbor classification. Technol Health Care 26(S1):509–519
27. Kubat M, Flotzinger D, Pfurtscheller G (1993) Discovering patterns in EEG-signals: comparative study of a few methods. In: 6th European conference on machine learning. pp 366–371. ECML'1993, Springer, Vienna, Austria, Apr 1993
28. Rupp R, Kleih SC, Leeb R, Millan J del R, Kübler A, Müller-Putz GR (2014) Brain-computer interfaces and assistive technology, Springer, pp 7–38
29. Lotze M, Halsband U (2006) Motor imagery. J Physiol Paris 99(4):386–395
30. Asensio-Cubero J, Gan JQ, Palaniappan R (2013) Multiresolution analysis over simple graphs for brain computer interfaces. J Neural Eng 10(4):21–26
31. Ortega J, Kimovski D, Gan JQ, Ortiz A, Damas M (2017) A parallel island approach to multiobjective feature selection for brain-computer interfaces. In: 14th international work-conference on artificial neural networks. pp 16–27. IWANN'2017, Springer, Cádiz, Spain, Jun 2017
32. Ortega J, Asensio-Cubero J, Gan JQ, Ortiz A (2015) Evolutionary multiobjective feature selection in multiresolution analysis for BCI. In: 3rd international conference on bioinformatics and biomedical engineering. pp 347–359. IWBBIO'2015, Springer, Granada, Spain, Apr 2015
33. Martín-Smith P, Ortega J, Asensio-Cubero J, Gan JQ, Ortiz A (2015) A label-aided filter method for multi-objective feature selection in EEG classification for BCI. In: 14th international work-conference on artificial neural networks. pp 133–144. IWANN'2015, Springer, Palma de Mallorca, Spain, Jun 2015
34. Martín-Smith P, Ortega J, Asensio-Cubero J, Gan JQ, Ortiz A (2017) A supervised filter method for multi-objective feature selection in EEG classification based on multi-resolution analysis for BCI. Neurocomputing 250:45–56
35. Ortega J, Asensio-Cubero J, Gan JQ, Ortiz A (2016) Classification of motor imagery tasks for BCI with multiresolution analysis and multiobjective feature selection. BioMedical Eng OnLine 15(1):149–164
36. González J, Ortega J, Escobar JJ, Damas M (2021) A lexicographic cooperative co-evolutionary approach for feature selection. Neurocomputing 463:59–76
37. Ortega J, Ortiz A, Martín-Smith P, Gan JQ, González J (2017) Deep belief networks and multiobjective feature selection for BCI with multiresolution analysis. In: 14th international work-conference on artificial neural networks. pp 28–39. IWANN'2017, Springer, Cádiz, Spain, Jun 2017
38. León J, Escobar JJ, Ortiz A, Ortega J, González J, Martín-Smith P, Gan JQ, Damas M (2020) Deep learning for eeg-based motor imagery classification: accuracy-cost trade-off. PLoS ONE 15(6):e0234178
39. Aquino-Brítez D, Ortiz A, Ortega J, León J, Formoso MA, Gan JQ, Escobar JJ (2021) Optimization of deep architectures for eeg signal classification: An automl approach using evolutionary algorithms. Sensors 21(6):2096
40. Lawhern VJ, Solon AJ, Waytowich NR, Gordon SM, Hung CP, Lance BJ (2018) EEGNet: A compact convolutional neural network for EEG-based brain-computer interfaces. J Neural Eng 15(5):e056013
41. Schirrmeister RT, Springenberg JT, Fiederer LDJ, Glasstetter M, Eggensperger K, Tangermann M, Hutter F, Burgard W, Ball T (2017) Deep learning with convolutional neural networks for EEG decoding and visualization. Hum Brain Mapp 38(11):5391–5420
42. Khan J, Bhatti MH, Khan UG, Iqbal R (2019) Multiclass EEG motor-imagery classification with sub-band common spatial patterns. J Wirel Commun Netw 174
43. Ding C, Peng H (2003) 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

44. Jo I, Lee S, Oh S (2019) Improved measures of redundancy and relevance for mRMR feature selection. Computers 8(2):42
45. Deng Z, Zhu X, Cheng D, Zong M, Zhang S (2016) Efficient kNN classification algorithm for big data. Neurocomputing 195:143–148
46. Raudys SJ, Jain AK (1991) Small sample size effects in statistical pattern recognition: Recommendations for practitioners. IEEE Trans Pattern Anal Mach Intell 13(3):252–264
47. Escobar JJ, Ortega J, González J, Damas M (2016) Improving memory accesses for heterogeneous parallel multi-objective feature selection on EEG classification. In: 4th international workshop on parallelism in bioinformatics. pp 372–383. PBIO'2016, Springer, Grenoble, France, Aug 2016
48. Luz Precio de la (2022) Public api for the pvpc regulated tariff of the spanish electricity market. https://api.preciodelaluz.org. Accessed 18 Nov 2022
49. Kale V, Gropp W (2010) Load balancing for regular meshes on SMPs with MPI. In: 7th European MPI User's group meeting. pp 229–238. EuroMPI'2010, Springer, Stuttgart, Germany, Sep 2010
50. Ding F, Wienke S, Zhang R (2015) Dynamic MPI parallel task scheduling based on a master-worker pattern in cloud computing. Int J Auton Adapt Commun Syst 8(4):424–438
51. Gvozdetska N, Globa L, Prokopets V (2019) 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
52. Babiuch M, Foltýnek P, Smutný P (2019) Using the ESP32 microcontroller for data processing. In: 20th international conference on carpathian control. pp 1–6. ICCC'2019, IEEE, Kraków-Wieliczka, Poland, May 2019
53. Judith AM, Priya SB, Mahendran RK, Gadekallu TR, Ambati LS (2022) Two-phase classification: ANN and A-SVM classifiers on motor imagery BCI. Asian J Control. https://doi.org/10.1002/asjc.2983
54. Dong Y, Chen J, Yang X, Deng L, Zhang X (2008) 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
55. Hassaballah M, Omran S, Mahdy YB (2008) A review of SIMD multimedia extensions and their usage in scientific and engineering applications. Comput J 51(6):630–649
56. Tripathi R, Sivaraman V, Tamarapalli V (2021) Distributed cost-aware fault-tolerant load balancing in geo-distributed data centers. IEEE Trans Green Commun Netw 6(1):472–483

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Authors and Affiliations

Juan José Escobar[1] · **Francisco Rodríguez**[2] · **Beatriz Prieto**[2] · **Dragi Kimovski**[3] · **Andrés Ortiz**[4] · **Miguel Damas**[2]

✉ Juan José Escobar
jjescobar@ugr.es

Francisco Rodríguez
cazz@correo.ugr.es

Beatriz Prieto
beap@ugr.es

Dragi Kimovski
Dragi.Kimovski@aau.at

Andrés Ortiz
aortiz@ic.uma.es

Miguel Damas
mdamas@ugr.es

[1] Department of Software Engineering, CITIC, University of Granada, Granada, Spain

[2] Department of Computer Engineering, Automation and Robotics, CITIC, University of Granada, Granada, Spain

[3] Institute of Information Technology, University of Klagenfurt, Klagenfurt, Austria

[4] Department of Communications Engineering, University of Málaga, Málaga, Spain