ARTICLE TEMPLATE

Block Size, Parallelism and Predictive Performance: Finding the Sweet Spot in Distributed Learning

Filipe Oliveira[a], Davide Carneiro[a], Miguel Guimarães[a], Óscar Oliveira[a] and Paulo Novais[b]

[a]CIICESI, ESTG, Politécnico do Porto, Portugal
[b]ALGORITMI Research Centre/LASI, University of Minho, Portugal

ARTICLE HISTORY
Compiled June 26, 2023

ABSTRACT
As distributed and multi-organization Machine Learning emerges, new challenges must be solved, such as diverse and low-quality data or real-time delivery. In this paper, we use a distributed learning environment to analyze the relationship between block size, parallelism, and predictor quality. Specifically, the goal is to find the optimum block size and the best heuristic to create distributed Ensembles. We evaluated three different heuristics and five block sizes on four publicly available datasets. Results show that using fewer but better base models matches or outperforms a standard Random Forest, and that 32 MB is the best block size.

KEYWORDS
Distributed Machine Learning; Distributed File System; Hadoop; Machine Learning

1. Introduction

The field of Machine Learning (ML) [1] has seen many advancements, especially in terms of algorithms [2–4], but new challenges continue to arise [5,6]. These challenges stem primarily from the large volume and diversity of data in current ML problems and the need for the real-time delivery of services based on that data [7]. This has led to the emergence of new fields of research based on streaming data, such as streaming analytics [8] and streaming ML [9].

Indeed, the proliferation of digital data in recent years has given rise to the need for ML algorithms that can process and analyze these large volumes of data in real-time. However, the challenges associated with handling such large and diverse datasets are significant. One of the primary challenges is the need for powerful computing resources to process and store the data. With the advent of big data, traditional relational databases are no longer sufficient to handle the sheer volume and complexity of the data. Therefore, ML practitioners have had to adopt new approaches, such as distributed computing and cloud-based storage solutions, to handle these large datasets [10].

Another challenge is the diversity of the data itself [11]. Large datasets can come from a wide range of sources, and the data can vary significantly in terms of quality,

CONTACT D. R. Carneiro. Email: dcarneiro@estg.ipp.pt

structure, and format. This makes it difficult to develop ML models that can accurately analyze and make predictions based on this data. Furthermore, data quality issues such as missing or erroneous data can lead to inaccurate results and false conclusions.

The need for real-time delivery of services based on data is another challenge. Real-time processing of data requires fast and efficient algorithms that can quickly analyze large volumes of data and make predictions in real-time. Furthermore, the results of these analyzes need to be delivered to users in real-time, which requires fast and reliable communication networks and the ability to handle large volumes of traffic.

In summary, the challenges associated with large volumes and diversity of data in ML problems and the need for real-time delivery of services based on that data require advanced computing resources, innovative approaches to data management, and efficient algorithms that can quickly and accurately analyze data in real-time. ML practitioners need to continually refine their techniques and adopt new approaches to stay ahead of these challenges and continue to drive innovation in this field.

Independently of the challenge, however, all the solutions have one characteristic in common: they are distributed. This means not only distributed storage [12] but also distributed processing [13] namely in terms of model training and serving [14]. Intuitively, while such systems allow working with larger volumes of data that would not fit in the memory or storage disc of a monolithic system, they also imply an increased effort in terms of coordination and synchronisation. This effort is not only computational but also often in terms of the complexity of the actual algorithms used, which must be adapted or developed anew.

Some of the relevant solutions (addressed in Section 2) include concepts such as Ensemble Learning [15], Distributed Learning, or Federated Learning. These concepts come together in the Continuously Evolving Distributed Ensembles (CEDEs) project, presented in this paper. CEDEs, however, goes further than a simple Ensemble, as it allows for Ensembles to evolve over time effortlessly, in computational terms, by not requiring a full re-training as data change. Moreover, Ensembles are seen as logic constructs, so they can easily be set up depending on the requirements of the user and on the state of the cluster.

Specifically, in this paper, we analyze the interesting relationship that emerged in CEDEs between data block size, degree of parallelism, and its impact on the quality of the Ensemble (This relationship is described in Section 3). One of our objectives is to find the optimum point at the intersection of these conflicting requirements. Indeed, block size influences all others in different ways. A smaller block size will increase the degree of parallelism, with a cost in terms of coordination overhead, and vice-versa. The impact on model quality is not so clear and will be studied. For instance, very small block sizes will allow for very dynamic and heterogeneous Ensembles, but the base models might be so poor (for the lack of data) that the whole Ensemble is weak in terms of predictive performance. On the other end, having very large block sizes might ultimately defeat the purpose of an Ensemble.

From the perspective of CEDEs and similar distributed learning systems, it is fundamental to find the optimal point so that a more efficient management of cluster resources can be implemented [16].

The remainder of this paper is organized as follows. Section 2 presents the related work, while Sections 3 and 4, present, respectively, the CEDES project and the problem statement. The datasets used and the methodology followed for validating the proposed approach are addressed in Section 5. The results are presented in Section 6 and discussed, together with some closing remarks, in Section 7.

2. Related Work

The new research challenges in the field of ML, addressed in the previous section, have been tackled by researchers from very different fields, using different approaches. However, to the extent of our knowledge, no author has looked specifically at how the issues of block size, degree of parallelism and predictive performance interact. This section deals with the related work in terms of different algorithms and approaches, both commercial and from research, to deal with the challenges of distributed ML.

Several efforts have been performed to devise parallel or distributed connectionist algorithms (such as Neural Networks or Deep Learning) to decrease their training time with large datasets - one of its main drawbacks. In these cases, the challenge is to maintain convergence and efficiency at scale. In [17], for instance, the authors propose the Kroneckerfactored Approximate Curvature (K-FAC) as an approximation of the Fisher Information Matrix that can be used in natural gradient optimizers, validating it with interesting results.

Other approaches consist in creating so-called Ensembles [18]. An Ensemble is a group of simpler models, also called base models (or weak learners), in which each one is trained with a part of the data. While each base model is usually a poor one if considered in isolation, grouping them under an Ensemble through some heuristic frequently holds better results than using a single complex model.

Different types of Ensemble exist, some of which are quite straightforward to implement in a distributed manner. Bagging Ensembles [19] are made up of a group of base models trained with different parts of the data selected at random. Depending on the implementation, this might mean that each base model sees a different subset of the rows and eventually even a different subset of the columns. For these reasons, Bagging Ensembles are very resilient to overfitting [20]. When it comes to computing predictions, in a Bagging Ensemble, each base model has the same weight in computing the final prediction, be it a regression (average prediction) or classification (most voted class) task, irrespective of the quality of the base model. Due to its nature, a distributed implementation of Bagging Ensembles comes naturally, as different base models can be trained on data selected from different parts of the dataset.

In a Stacking Ensemble [21], on the other hand, many different models are trained *on the same data*. This, in itself, makes it harder to distribute the algorithm if the data is distributed. In short, the algorithms used to train the models must be able to learn in a distributed manner. As in Bagging Ensembles, some of these models will be better than others. However, here, each base model might have a different weight when making predictions, or some base models might not be considered at all. This is determined in the final stage of the training of the Ensemble, in which an additional model is trained to learn how to best combine the predictions.

Finally, Boosting Ensembles [22] considers that base models are added sequentially, in which each new base model attempts to minimize the error on those instances that were more poorly classified by the previously trained one. The fact that there is an active focus on adapting to previous errors makes Boosting Ensembles more prone to overfitting, especially in scenarios in which there are outliers. Since models are trained sequentially, this is also the type of Ensemble that is less prone to be distributed or parallelized.

There are thus different possible approaches, each of which with its advantages and disadvantages. The best one can only be identified on a case-by-case basis [22]. However, in terms of their adequacy to be used as a solution for distributed learning, an ordering from most suited (Bagging) to least suited (Stacking) can be put forward.

In the last years, another layer of challenges became evident when it comes to distributed learning. Data sources and storage grew so much in volume that they started to exist over or represent different regions, with eventually different legislation in what concerns privacy and data protection. This led to the emergence of Federated Learning [23] which allows for organizations, countries or regions to collaboratively train ML models, in a distributed manner, while still respecting and preserving data ownership and privacy. Data is kept decentralized, with their respective owners, while models are trained with those data and shared.

Asides from cloud-based solutions, there are also open-source and other commercial ML tools that address these challenges. One such example is H2O: a ML platform that offers open-source tools and libraries to help data scientists and engineers create and implement ML models. Its purpose is to simplify the process of working with large datasets and algorithms.

H2O has a significant advantage due to its capability to distribute its workload across multiple machines. This feature enables H2O to process data at scale, a crucial requirement for ML tasks that involve processing large datasets. H2O utilises the H2O cluster, a distributed system that enables users to train and deploy ML models in parallel using multiple nodes. In addiction to its ability to distribute workloads, H2O's intuitive interface makes it user-friendly. Its interface is simple and straightforward, allowing data scientists to easily create and deploy ML models. H2O also provides a broad spectrum of ML algorithms such as Deep Learning, Gradient Boosting, and Generalised Modelling, to name a few. To begin working with H2O in a distributed way, the first step is to initialise the cluster, either locally or on a remote machine. After starting the cluster, data can be uploaded from a disk into H2O Frames, which are distributed data frames. H2O was designed to process big data in a way similar to platforms like Hadoop and Spark, which were popular when H2O was first introduced. It also uses a lock-free distributed key/value store [24] internally to read and write data (models, frames, objects) asynchronously across all nodes and machines. The architecture of the H2O cluster is similar to a Hadoop Distributed File System (HDFS) architecture, with a coordinator node that manages the cluster and coordinates tasks among worker nodes, and worker nodes that are responsible for performing all of the computation required to respond to tasks determined by the coordinator. Overall, the H2O architecture is designed to be highly scalable, fault-tolerant, and efficient in processing large datasets for ML tasks, and can be used by anyone, from a single user working on a standard laptop to groups of users working on large distributed clusters.

As this section shows, the approaches to deal with the new challenges of ML are many. Moreover, in some cases, the configuration of the data cluster (such as the block size) may have an impact not only on the performance of the storage but also on the performance of the predictive model. To the extent of our knowledge, this issue has never been investigated before, and the rest of the paper addresses the relevance of this issue and the conclusions achieved from investigating it.

3. The CEDEs project

This section describes, from a functional and high-level perspective, the Continuously Evolving Distributed Ensembles (CEDEs) project. The main objective of CEDEs is to create an ecosystem for the distributed training of ML models, allowing the models to adapt as data changes by adding new base models as new blocks are completed, and eventually removing other base models, in a cost-effective way. This addresses not

only the challenge of handling large datasets but also the capability of continuously learning from streaming data.

CEDEs takes advantage of existing block-based distributed file systems, such as HDFS [25] to parallelize and distribute learning tasks efficiently by following the principle of data locality [26] which means that the computation is brought to where the data is, instead of the other way around. Instead of conventional models, CEDEs uses Ensembles [18], where a base model is trained for a specific block of a dataset and Ensembles are built in real-time by combining available models based on performance or other criteria. This is managed by an Optimization module that adapts the Ensemble as data changes.

The system incorporates two additional mechanisms: replication and balancing. Replication enables the automatic creation of block replicas so that the same block is available in multiple locations of the cluster simultaneously. This mechanism increases the use of storage space but makes it easier for the Optimization module (discussed in Section 3.2) to find suitable nodes to carry out tasks. Balancing, on the other hand, distributes the blocks evenly across the cluster, ensuring that the load is distributed appropriately when executing tasks.

Finally, CEDEs also stores the base models in the distributed file system and replicates them. This means that not only the Ensemble itself is distributed, which speeds up predictions, but also that, thanks to replication, multiple nodes will have the same base models available for making predictions. The optimization module is responsible for deciding which base models to use and on which nodes to make predictions.

The remaining of this section describes the data model and the architecture of CEDEs, while Section 5 details the methodology followed to validate it.

3.1. Data Model

In what concerns the data model, there are seven main entities as depicted in Figure 1. One of the central entities is the *Organization*. Multiple organizations are meant to share the same instance of CEDEs. Moreover, ideally, each organization contributes with its own hardware, i.e., servers that are added to the cluster and in which the data of each organization are stored.

In line with the Federated Learning concept, each instance of CEDEs is meant to exist in a specific ML domain (e.g. Fraud Detection, Healthcare, Finance). However, datasets and organizations of multiple domains can obviously coexist. The system allows the existence of multiple organizations as a part of an instance, so they are expected to be, in some way, related to the domain of that instance.

The participation of the organizations can be very different, depending on how they use the available services. For example, an organization can be mostly a data creator, contributing towards the availability of more models and, consequently, enriching the system. On the other hand, another organization might not have the ability to produce data but might make heavy use of existing models for predictive purposes, such as would happen in a newly created organization. Organizations located at any point between these two extremes may exist.

Every action organizations perform (e.g., add data, train model, request predictions) is stored in a shared blockchain, and imbalances such as those described above might be balanced through contributions towards the ecosystem (e.g., buying and selling credits). Although this aspect is not the focus of this paper, it is described in more detail below. In summary, the goal is to have a data, model and hardware ecosystem

in which organizations receive credits whenever their shared hardware or models are used by others and pay credits whenever they want to use others' models.

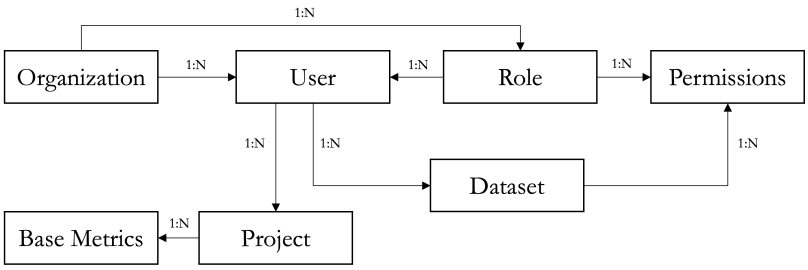


Figure 1. Simplified view of the data model that supports the proposed system.

The central entity in the data model is the *User*. Each organization may hold multiple users. Users have the ability to create and edit ML projects, and import or access datasets. *Projects* and *Datasets* are thus another two important entities in the data model.

The access to each dataset is governed by a system of *Roles* and *Permissions*. Organizations can establish various roles, such as Data Scientist, ML Engineer, and Data Engineer, and assign them to their users. Permissions are then granted based on the roles of the users. Furthermore, when a user uploads a new dataset, they become the owner of that dataset, which is crucial for determining the provenance of the data.

The information that the user provides when training a project is stored in the *Project* entity. This includes, among others, the name and general description of the project, the desired dataset, which algorithm(s) is/are to be used in the training process and respective configurations (optional), etc.

In order to give users the ability to create heterogeneous Ensembles, as described in Section 5, the system allows multiple algorithms to be used in a single project. In this case, the user must provide the relative proportion of each algorithm (e.g., 50/30/20), so that each base model can be submitted to these probabilities in the training process.

Using the proportions above means that each base model has 50% chance to be trained with algorithm a, 30% with algorithm b and 20% with algorithm c. This means that CEDEs supports both homogeneous and heterogeneous Ensembles.

The user may also decide between using a standard configuration or fine-tuning the hyper-parameters of each algorithm used. This functionality is aimed at users who are more familiar with ML and want to use their additional knowledge to create their own customized models. The configuration depends on the type of each algorithm. For example, a Decision Tree may be configured in terms of maximum depth or minimum number of instances per leaf (among others), while a neural network may be configured in terms of activation function or the number of layers and respective neurons (among others).

Finally, each project has an associated *Base Metrics* entity. This holds the performance metrics (e.g. RMSE, MAE, Accuracy, F1-Score) [27] of each base model trained. Base metrics are also used to compute the performance metrics of a given Ensemble, based on the metrics of the base models that make up the Ensemble, which are averaged.

3.2. Architecture

The architecture of the suggested system comprises several key components, as outlined in this section and depicted in Figure 2. In its present form, CEDEs is distributed as a Docker application, which can be easily deployed. In its implementation we do not use the official Docker images for Hadoop, Spark or any of the other software used. Instead, the containers are all built based on a vanilla `ubuntu:bionic` image, in which all the relevant software is installed and configured automatically, through a Dockerfile. This includes installing and configuring the HDFS in cluster mode, SSH access for the containers, setting up environment variables, installing necessary packages or formatting the distributed file system. This makes the process very similar to what has to be done when installing it in a cluster with several physical machines, and the Dockerfile can even be used as a manual for deploying CEDEs. The number of containers (1 coordinator + n workers) can easily be set up, to test clusters with different sizes. This approach was followed given the lack of an appropriate physical cluster, but having as a requirement that the transition to a physical cluster will be as smooth as possible.

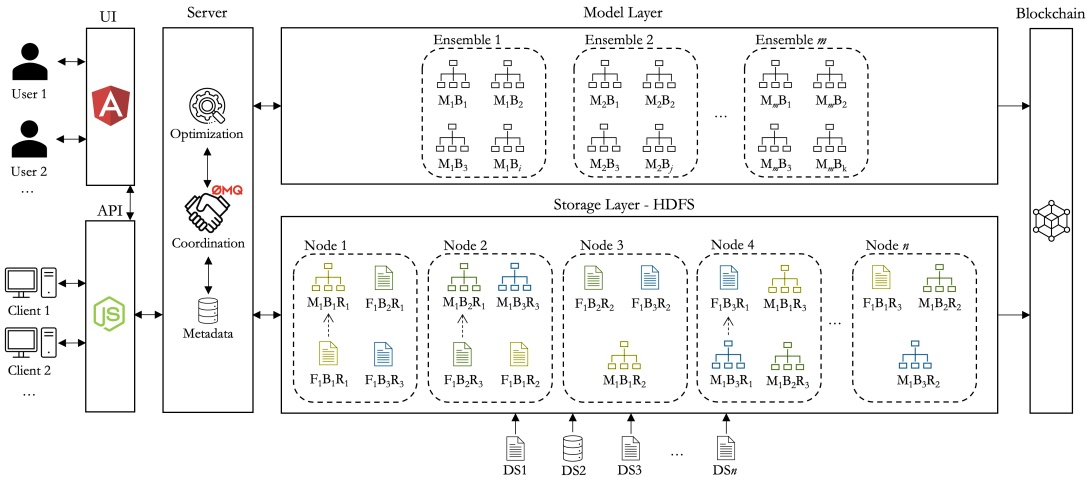


Figure 2. Depiction of the proposed architecture with a possible sample scenario.

The central component of the architecture is the Storage Layer which is implemented as an HDFS cluster. In this layer, large amounts of data are divided into smaller segments called blocks, each with a fixed size (generally 128MB). This layer can handle various data sources, including different formats (e.g., CSV, Parquet, Avro) and databases. The process begins with a user uploading a data source to the HDFS, which is then split, distributed, and replicated across the cluster.

Replication is controlled by a so-called replication factor (usually with a value of 3). This means that any data stored in the file system, which includes both raw data and serialized base models, will be available simultaneously in several nodes. This increases the availability of these elements, namely of candidate nodes to train base models, at the cost of some storage overhead.

Once the upload process is completed, the dataset is available for use in a new ML project. The user now is able to create a new project, select the desired dataset, and provide the necessary information as previously described. Once the project configuration is complete, the user can begin training a new Ensemble. This process is managed by the Coordination module, which interacts with the Optimization and Metadata

modules.

The training of an Ensemble occurs in a distributed manner and takes advantage of two factors: the replication and availability of blocks across multiple nodes and the principle of data locality. Since the dataset is divided into blocks, a base model is trained for each block of a file (excluding replicas), which, when combined, form the Ensemble. As a result, for each base model that must be trained, the Optimization module must decide which of the candidate nodes (where the corresponding blocks are located) will be selected. The goal of this optimization is to minimize the makespan. Further details on the implementation of this module are provided in [28]

The inputs used by the Optimization and the Coordination modules to assign tasks are the following:

Block location - Taking into account that one of this system's pillars is data locality, the Coordination module needs to know what blocks are part of each dataset, and where each of them and their respective replicas are located and available at the moment of training so that computation (e.g., base model training) can be moved to where the data resides. The replication factor is the number that determines how many replicas for each block are created and distributed across the cluster (which is typically 3 but is dependent on the HDFS configuration and easily manipulated). One base model is trained for each block of data, resulting in a 1:1 relationship between the blocks of a specific dataset and the base models that make up an ensemble of a specific ML project that utilizes that dataset. This information is obtained from the Metadata module.

Task cost prediction - In order to get an estimation of the cost of each individual task (e.g., training of each base model) an approach based on meta-learning was implemented. Based on data collected from a large number of ML problems and datasets obtained over time, this approach is able to accurately estimate the training time of a given algorithm/configuration for a dataset with given characteristics (meta-features). This approach, which provides fairly good results in terms of task complexity/cost predictions, is further described in [29,30]. This information is obtained from the Optimization module.

Nodes state - Another relevant input for the Optimization module to assign tasks is the state of each node in the cluster. The Optimization module uses several forms to get those metrics, such as Hadoop's health checker service which is able to determine if the node is healthy or dead, and services that return the current load of the nodes (e.g., CPU, memory, disk). The features such as the cost and size of each node's task queue are maintained and made available by the Coordination module, which knows, at each moment, the pending tasks in each node and their characteristics.

Once the Coordination module receives the information relative to the best candidate where to allocate each task of a specific ML project, the training process begins. The process starts with the coordinator node broadcasting the task information to all of the worker nodes in the cluster. This communication process is implemented with ZeroMQ: a brokerless asynchronous messaging library that implements multiple socket communication patterns that are useful for implementing distributed systems.

The base models that are generated in the training process are serialized and stored in HDFS as the workers complete the respective training and consequently replicated and distributed across the cluster due to replication and balancing factors. The user is able to access this information in real-time in the client application. As the worker nodes finish the training of all the models that they have been assigned, the coordinator is notified and the worker can move on to the next task in the queue if there is one.

This whole process is illustrated in Figure 2, in which the following terminology is used: $F_iB_jR_k$ represents a replica k of block j of file i. The same logic is used for models,

which start with the letter M instead. An arrow between a specific block replica and a model means that a base model was trained from that block on that node. Replicas of that base model may however exist in other nodes, created through the Hadoop replication mechanism after the base model was stored in the HDFS.

When a client makes a request for predictions for a given project, a process identical to the training process takes place. Initially, the Coordination module requests information regarding which of the base models will constitute the predictive Ensemble. This is determined by the Optimization module following a specific heuristic. This means that the Ensemble is created dynamically in real-time, based on the available base models and/or their number and quality, depending on the heuristic used. The process is efficient in the sense that all base models are already trained and available, and constitute the pool of candidate models. The process of constituting an Ensemble is thus an optimization problem that only finds the best combination of models, as described before, based on their characteristics. A previous version of this mechanism was implemented using Genetic Algorithms as described in [31]. Nevertheless, a more efficient approach is now being implemented.

As base models start making their predictions, in a distributed manner, they forward them to the Coordinator. The predictions are then combined by the Coordinator according to the selected heuristic, and stored in the HDFS. Finally, when this process ends, the user is notified. As described in Section 5, different heuristics are available.

The Blockchain module records all user or client activity within the system, such as data importation, model training requests, prediction requests, etc. The API module encompasses all the necessary endpoints for interacting with the system's services and communicates with the Coordination module through ZeroMQ messages, as needed. External client applications can utilize this API. There's also a User Interface designed for human users, which communicates with the aforementioned API.

The implementation of the entire system is still in progress, with the focus currently being on the Optimization module.

4. Problem Statement

The implementation of CEDEs, a project of exploratory nature, highlighted some of the interesting conflicts that exist in distributed systems. These often stem either from configuration options of the tools used or from software implementation decisions. This paper focuses in one of these conflicts, and this section describes the methodology followed to try to find an answer to the open questions.

The main issue is that of block size, which is a configuration of the distributed file system used. In HDFS, the block size affects the way a file is stored. A small block size will result in smaller blocks, which will be quicker to balance and replicate across the cluster. However, it will also have a significant overhead in terms of management of the file system as the meta-data stored for each block is independent of the block size. So, the smaller the block size is, the larger the number of blocks that must be maintained, hence the overhead. Indeed, the main limitation of HDFS is memory: it performs better with a few large files than with many small ones.

The implications for CEDEs, however, go much further. Given that there is a 1:1 relationship between blocks and base models, a smaller block size means that there will be more blocks, hence more base models. Base models will also in principle be smaller and simpler, and also in principle weaker in predictive terms since they were trained with fewer data. Given that the performance of an Ensemble is measured from

the average performance of its base models, if models are generally weaker, so will the Ensembles created. While up to a certain point, a smaller block size increases parallelism, as there are more base models that can be trained in parallel and their cost to train is smaller, it also increases the overhead related to coordination and task distribution.

On the other end, i.e., if a large block size is used, the performance of the base models and the Ensemble is in principle improved, the coordination overhead is minimized, but the degree of parallelism decreases. Moreover, if the block size is large enough that there are only a few blocks, the advantages of Ensemble methods are lost and, ultimately, we might have an Ensemble with a single model.

One of the objectives of this work is to present and discuss this problem, first, and then to find the optimal point that maximizes both parallelism and model quality. Evidently, this analyzis should be done on a case-by-case basis, as it depends on the number of available nodes in the cluster. In the following Section, we perform it in a generic manner.

Another goal is to evaluate different heuristics for creating Ensembles, which is also done in the following Section. Specifically, we evaluate and compare three possible heuristics. As they are available to be chosen by the user in CEDEs when configuring the ML project, it is important to compare them in terms of performance.

5. Materials and Methods

The previous section described the main problem addressed by this work. This section details the methodology followed in its analyzis (Figure 3). Four different publicly available datasets were used in the experiments conducted. A characterization of these datasets is provided in Table 1.

Dataset	**Rows**	**Columns**	**Size (Mb)**
Covid infections	4623663	19	585
Solar production	2782080	15	399
Temperature	2434608	20	351
Car prices	1957675	1108	4160

Table 1. Characterization of the datasets used.

The original datasets were obtained and pre-processed. This included, among other tasks, encoding certain variables or removing outliers. Outliers were removed from the original dataset using the 1.5 × IQR rule. The interquartile range (IQR) is obtained from Equation (1), in which q_{75} and q_{25} represent the first and third quartile, respectively.

$$iqr = q_{75} - q_{25} \tag{1}$$

Then, the lower and upper limits for removing outliers are given by Equations (2) and (3), respectively.

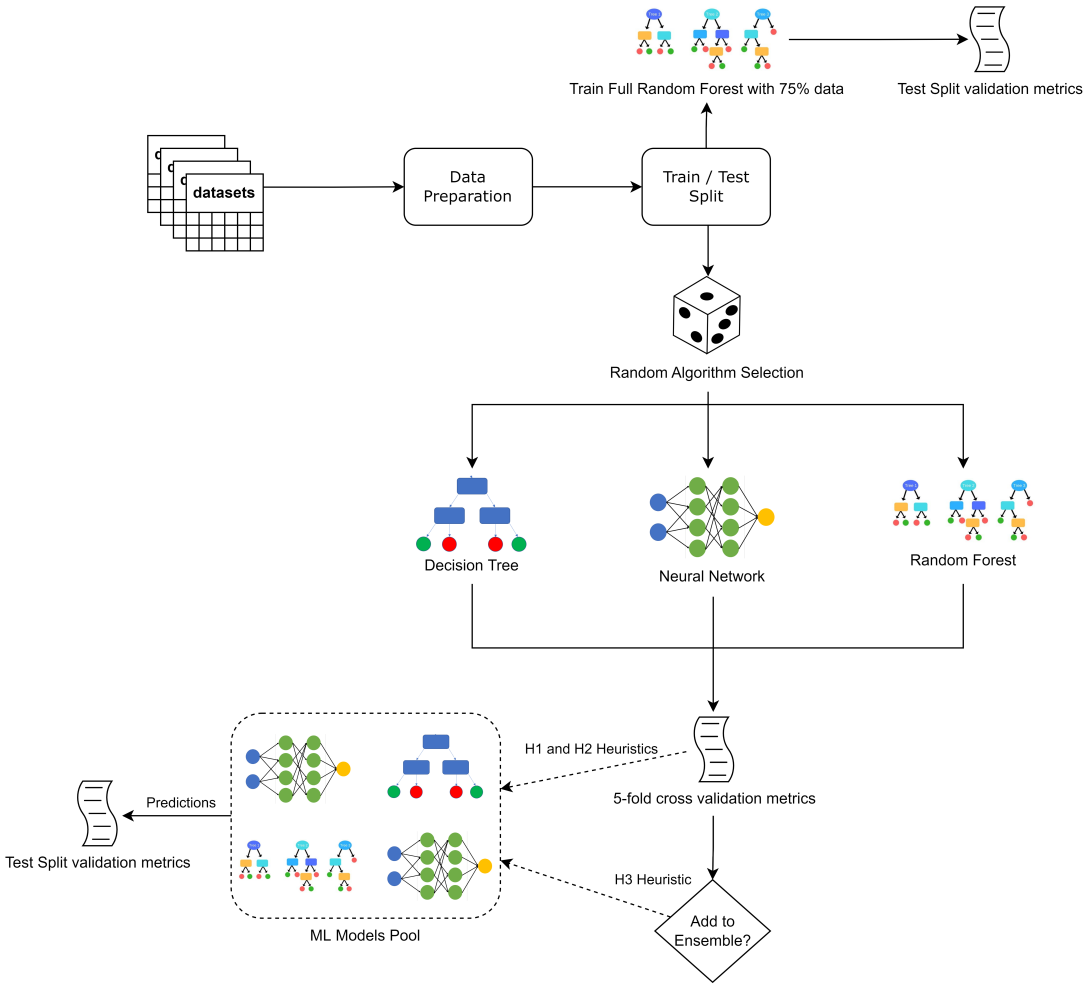


Figure 3. Methodology followed in each of the experiments.

$$lower = q_{25} - 1.5 \times iqr \tag{2}$$

$$upper = q_{75} + 1.5 \times iqr \tag{3}$$

Next, each dataset is filtered by the dependent variable, as given by Equation (4) in which x denotes each instance of dataset X.

$$filtered = \{x \in X \mid x < upper \ \wedge \ x > lower\} \tag{4}$$

These processed datasets were uploaded to the HDFS. Then, a series of ML experiments were conducted as follows. The datasets were first shuffled and then split into train and test subsets using a 75/25 ratio. The train sets were used for training the different Ensembles while the test sets, which were held out, were only used for testing

the Ensembles and their ability to generalize.

Over the experiments, the HDFS was successively configured with different block sizes $B = [4, 8, 16, 32, 64]$ (in Mb). This resulted in each data set being split into different numbers of blocks depending on their size.

For each block size and each dataset, three different heuristics for building Ensembles were tested, deemed H_1 to H_3, detailed further below. This means that a total of 60 Ensembles were trained: 5 block sizes $\times$ 4 datasets $\times$ 3 heuristics. These Ensembles have different numbers of base models, depending on the block size.

The algorithm for each base model was selected randomly, and the standard configurations described in Table 2 were used. In the case of Neural Networks, the use of the different activation functions was also selected randomly. This means, on the one hand, that the Ensembles created are heterogeneous and, on the other hand, that if the process is repeated the results might vary slightly, as the choice of the algorithms is random. However, given the size of the data, the variation should not be significant.

CEDEs uses the scikit-learn library. Specifically, in this study, three algorithms were used: Random Forest (sklearn.Ensemble. RandomForestClassifier), Decision Tree (sklearn.tree.DecisionTreeClassifier) and Neural Network (sklearn.neural_network.MLPClassifier). The three tested heuristics work as follows:

- H_1 - The Ensemble is created following a Bagging approach, with all the base models being considered. However, each base model has a weight that is inversely proportional to its quality, given by the Root Mean Square Error (RMSE) calculated through 5-fold cross-validation.
- H_2 - The Ensemble is created following a traditional Bagging approach, with all the base models being used, and each having equal weight.
- H_3 - The Ensemble is created by selecting only the top 50% base models, and with all selected models having the same weight. Models are ranked by their RMSE, calculated through 5-fold cross-validation.

The mathematical formula for calculating the RMSE (see [32]) is:

$$\text{RMSE}(y, \hat{y}) = \sqrt{\frac{\sum_{i=0}^{N-1}(y_i - \hat{y}_i)^2}{N}} \tag{5}$$

In H_1, the weight of a given model m in an Ensemble of n models in which ϵ_i represents the error metric (RMSE) of model i is given by:

$$W_m = \frac{\sum_{i=1}^{n} \epsilon_i, i \neq m}{\sum_{i=1}^{n} \epsilon_i} \times \frac{1}{n-1} \tag{6}$$

In this process, as in the development of CEDEs, we assume that the data may not be evenly distributed, that is, there may be trends and fluctuations, and the items in different blocks (or in the same block for that matter) may be taken from different probability distributions. This may result in significant fluctuations in model performance across the base models of an Ensemble, depending on their input blocks, which are explored by the Optimization module, that may use different criteria to build the best possible Ensemble (e.g., using a subset of the best models).

We also assume that all the instances are from independent events:

Algorithm	Parameter	Value
Decision Tree	max_depth	25
	min_samples_split	2
	max_leaf_nodes	infinite
	ccp_alpha	0.5
Neural Network	hidden_layers_size	(100, 50, 20)
	activation	relu/tanh
	solver	adam
	alpha	0.0001
	learning_rate	constant
	max_iter	150
Random Forest	max_depth	10
	nr_estimators	10

Table 2. Configurations used to train the heterogeneous Ensembles (defined randomly).

$$\forall i \neq j \; p(x^{(i)}, x^{(j)}) = p(x^{(i)})p(x^{(j)}) \tag{7}$$

That is, there is no relationship between instances, including temporal/order relationships, while datasets such as these might be used in CEDEs, the algorithms implemented so far are not the most adequate for these tasks. In summary, CEDEs was designed and validated in scenarios of independent non-identically distributed data [33].

As the base models are trained, their quality is estimated by means of the RMSE, in two ways. First, the RMSE is calculated through 5-fold cross-validation, during the training process. Then, since there is a test set for each dataset, it is also calculated on the test set, to evaluate its ability to generalize.

This is also relevant to determine to which extent the cross-validation RMSE is a good indicator of future model quality, since in H_1 we are using this RMSE to obtain the weight of the model, and in H_3 to select the top 50% models. A strong correlation between both RMSEs will indicate that such assumption is valid, and that the cross-validation RMSE can be used as a reliable indicator of model quality. In a scenario in which the test data is, for some reason, very different from the training data, as with concept drift, this might not hold true. A low correlation would thus point that out.

For this reason, in the following section, we start by comparing the cross-validation RMSE with the RMSE on test data. Then, we analyze the values of RMSE of each block size and heuristic, to find the best configuration for CEDEs.

6. Results

As described in the previous section, we started by analyzing how the RMSE of the Ensembles calculated through cross-validation correlates with the RMSE calculated on

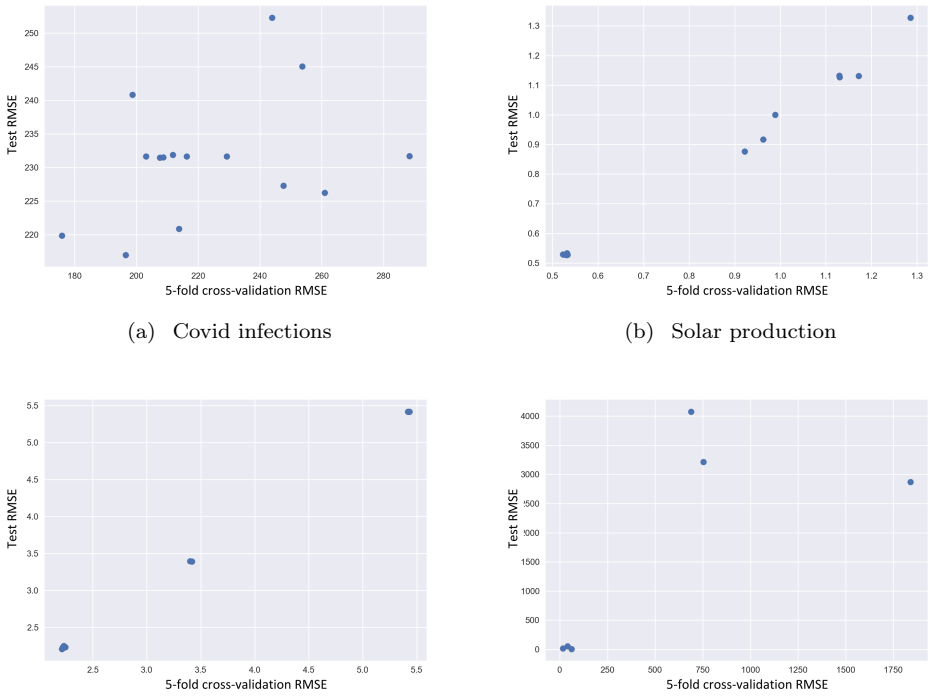
Dataset	4 MB	8 MB	16 MB	32 MB	64 MB
Covid infections	0.79	0.66	0.63	0.35	0.31
Solar production	0.93	0.94	0.99	0.99	0.98
Temperature	0.99	0.99	0.99	0.99	0.99
Car prices	0.29	0.25	0.43	0.71	0.99

Table 3. Correlations between the RMSE measured through 5-fold cross-validation and on the test data.

the test data. In this section, when individual values of RMSE are mentioned or analyzed (such as in the following figures) we show the absolute value of RMSE. However, when the performance of different models (for different datasets) is compared, we do it as a percentage of the range of values of the dependent variable. This is to remove the dependency of the RMSE on the scale of the dependent variable, so that a fair comparison between different ML problems can be done.

Two interesting trends emerge from the analyzis of Table 3. In the Covid infections dataset, the trend is for the correlation to be weaker as the block size increases, whereas in the remaining three datasets, the correlation generally gets stronger as the block size increases. The latter is the expected and intuitive behavior and it indicates that smaller block sizes might lead to base models whose performance, albeit good, might be misleading in the sense that it might not translate to new data. That is, models that do not generalize well.

Figure 4 depicts this information visually, for the 32 MB block size. Each dot represents one base model. The x-axis contains the RMSE measured through 5-fold cross-validation, while the y-axis contains the RMSE measured on the test set.



(a) Covid infections (b) Solar production

(c) Temperature (d) Car prices

Figure 4. RMSE measured through cross-validation vs. RMSE measured on the test dataset, for the 32MB block size. This allows to determine the extent to which the performance of base models (local), measured through cross-validation, is a good indicator of generalization (RMSE measured on the test set).

Asides from the Ensembles trained following the methodology described in the previous section, and in order to compare their performance with a baseline, one Random Forest was also trained for each dataset, using all of the data. The RMSE of the Random Forest was calculated through 5-fold cross-validation.

Since the RMSE depends on the scale of the dependent variable, models for different problems cannot be directly compared. For this reason, and in order to be able to draw general conclusions, we conduct an analyzis in terms of the RMSE divided by the range of the outlier limits of the dependent variable, calculated using the 1.5 IQR rule. This information will give the error as a percentage of the range of values.

Analyzing Tables 4-6, it can be generally concluded that as the block size increases, the error decreases. There is, however, the already discussed drawback that a larger block size also results in a lower level of parallelism, as there are fewer candidate nodes. Finding the optimum block size is paramount to maximize parallelism while minimizing error metrics.

It is clear that the Covid infections dataset is the most challenging one, both for CEDEs and for a standard Random Forest, due to the characteristics of the dataset: this is a time-series of historic data concerning Covid infections. Forecasting time-series data, which has specific components such as seasonality, trend, or cyclical variations, is challenging and usually requires more specific algorithms such as ARIMA or LSTM.

The different characteristics of this dataset are also visible when analyzing the correlations between the cross-validation and test errors. In all the other datasets, this correlation increases with the block size, as the more data there are in each block, the closer it will be to the full data or to the test data (better generalization of the model).

In the case of the Covid dataset, the opposite occurs.

In terms of model performance, the error of the baseline Random Forest is 6.68% (of the range of values of the dependent variable). In the case of the Ensembles trained, this error ranges from 7.4% (H_1, 64 MB) to 8.02% (H_3, 4 MB). The baseline Random Forest outperforms our Ensembles in every scenario, although the results are close.

The remaining datasets hold generally better results, both in the baseline Random Forest and in our Ensembles.

In the Solar production dataset, the baseline Random Forest has an error of 0.08%. Like with the Covid infections dataset, the baseline Random Forest outperforms our Ensembles in all scenarios tested. The best Ensemble for this dataset was obtained for H_3 and a block size of 32 MB (0.41%). The worst performance happens with H_2 and a block size of 16 MB (0.67%).

The situation is different in the other two scenarios, in which the proposed approach is able to outperform the baseline Random Forest.

In the Temperature dataset, the baseline Random Forest has an error of 0.078%. CEDEs outperformed this in all the 15 scenarios tested for this dataset. The best performance was observed at H_3 with a block size of 32 MB (0.022%). The worst performance was 0.0317% and was observed twice, in H_1 and H_2 with 4 MB.

Finally, when it comes to the Car prices dataset, the baseline Random Forest has an error of only 0.001%. The best Ensemble obtained by CEDEs is able to outperform it with an error of 0.0002%. This happens with H_3 with 32 MB. The worst performance was observed in H_2 with 4 MB (0.022%).

H_1 - Bagging Ensemble, base model weight inversely proportional to RMSE						
Dataset	**RF**	**4 MB**	**8 MB**	**16 MB**	**32 MB**	**64 MB**
Covid infections	**6,6893**	7,8571	7,7857	7,6500	7,6393	7,4000
Solar production	0,0787	0,5941	0,6353	0,6634	**0,5135**	0,5210
Temperature	0,0783	0,0317	0,0304	0,0313	0,0288	**0,0271**
Car prices	**0,0010**	0,0221	0,0197	0,0091	0,0167	0,0065

Table 4. RMSE in percentage of the scale of the dependent variable, for each block size and heuristic H_1.

H_2 - Bagging Ensemble, all base models have the same weight						
Dataset	**RF**	**4 MB**	**8 MB**	**16 MB**	**32 MB**	**64 MB**
Covid infections	**6,6893**	7,8571	7,7857	7,6464	7,6393	7,4036
Solar production	**0,0787**	0,5941	0,6372	0,6672	0,5229	0,5360
Temperature	0,0783	0,0317	0,0304	0,0313	0,0292	**0,0275**
Car prices	**0,0010**	0,0222	0,0197	0,0093	0,0193	0,0090

Table 5. RMSE in percentage of the scale of the dependent variable, for each block size and heuristic H_2.

Next, we present a visual analyzis of the performance of some of the Ensembles trained. Since H_3 is the best performing heuristic (i.e., is the best heuristic for the temperature and car prices datasets, even comparing to the baseline Random Forest) and to avoid making the paper too large, we restrict our analyzis to this heuristic.

Figures 5 to 8 depict the results for the Covid infections, Solar production, Temperature and Car prices datasets, respectively. The figures represent the observed values (x-axis) vs. the values predicted by each Ensemble (y-axis). In this case, the units of

H_3 - Ensemble with the top 50% base models (RMSE), all base models have the same weight						
Dataset	**RF**	**4 MB**	**8 MB**	**16 MB**	**32 MB**	**64 MB**
Covid infections	**6,6893**	8,0214	7,9143	7,7964	7,6571	7,4321
Solar production	**0,0787**	0,4591	0,4779	0,5135	0,4123	0,4160
Temperature	0,0783	0,0242	0,0233	0,0267	**0,0221**	0,0238
Car prices	0,0010	0,0101	0,0178	0,0100	**0,0002**	0,0061

Table 6. RMSE in percentage of the scale of the dependent variable, for each block size and heuristic H_3.

the axes are those of the original target variable. The diagonal represents the line of perfect prediction. That is, the closer the points are to the diagonal the better, meaning the predicted values are closer to the real ones.

As discussed before, the Covid infections dataset is by far the dataset in which results are worst. However, the same happens with the baseline Random Forest so this can be attributed to this being an intrinsically complex problem, or the variables present in the dataset being of little relevance.

In the rest of the datasets, the behavior of the Ensembles is much more satisfactory, especially in the Car prices dataset in which at 32 MB the predictions are nearly perfect, while in other block sizes, there seems to be a small bias towards underestimating the real value.

7. Discussion, Conclusions and Future Work

In this paper, we presented CEDEs - a distributed environment for Machine Learning applications. CEDEs has several innovative key aspects, one of the more relevant being that Ensembles can be assembled in real-time, based on the available base models, and on other requirements such as the state of the cluster. This not only makes it a very dynamic approach, but it also allows it to maintain Ensembles updated through time, by adding new base models and removing others. This is all done by the optimization module, which is still not fully developed yet.

The objective of this paper was focused on a more specific but relevant aspect: to study how block size and different heuristics for building the Ensemble affect its quality. Specifically, we wanted to determine if one particular heuristic appears to be generally better, as well as a particular block size.

To this end, three different heuristics were tested, which were described in Section 5. Concerning the implemented heuristics, H_3 appears to hold generally lower error rates. This is a heuristic that considers only the best 50% base models available (independently of their number) and in which each model has the same weight.

This is a promising result as it points out that smaller Ensembles might outperform more complex ones. A possible conclusion is that the worst-performing base models might be contributing negatively to the performance of the Ensemble. This will allow for the implementation of a more efficient management of the base models. Namely, those that are continuously left out of the Ensembles due to their poor performance might eventually be deleted and removed from the HDFS, thus freeing resources. The only scenario in which H_3 did not perform better was in the Covid infections dataset. However, this dataset is clearly a difficult one for the algorithms considered since it is a time-series dataset, so it should not cast doubt on the adequability of H_3.

Another interesting result is that in all scenarios, the 32 MB block size outperforms all other options. From the perspective of the development of CEDEs this is interesting as it also represents a good compromise between parallelism, coordination overhead and Ensemble size. That is, smaller block sizes would increase coordination overhead, but apparently they also have worse results, so they should be avoided. Larger block sizes would decrease this overhead, but the results also appear to worsen. So, from these results we conclude that 32MB is the optimal block size.

However, we would also like to point out that the goal of this work was not to achieve the best results possible for each dataset, but to find the best heuristic and block size. In fact, we used standard configurations for each algorithm, arbitrarily defined, and equal overall problems. Better results would no doubt be obtained if an effort was put into trying to find better configurations, through trial and error or using some heuristic such as a grid search.

Still, the results are considered satisfactory. In two out of four datasets (namely, Temperature and Car prices), CEDEs outperforms a standard Random Forest. In the other two datasets (Covid infections and Solar production) the results were very similar.

The use of Ensembles for implementing the heuristics has both advantages and disadvantages. While, on the one hand, it allows to efficiently adapt the model and minimizes the risk of overfitting, on the other hand it reduces the interpretability of the models, as these become more complex. Thus, they generally fail to provide insights into the underlying relationships between features and outcomes. In the future, we will include developments from the field of explainable AI [34] in CEDEs, which is currently one of the most relevant topics of research in AI, namely by integrating model-agnostic approaches such as LIME [35] or Shapley values [36].

Another potential issue, which will be addressed in future work, is bias detection. Indeed, Ensemble machine learning models are often used to improve the accuracy and robustness of predictions by combining the outputs of multiple base models. While they can be effective at reducing variance and improving generalization performance, they can also be susceptible to bias in certain cases.

One potential source of bias in ensemble models is the selection of base models. If the base models are all trained on similar data or use similar algorithms, they may all make similar errors or exhibit similar biases, which can be amplified when their outputs are combined. Therefore, it is important to ensure that the base models are diverse and complementary, and that their biases are not correlated. We are currently working at including each block's meta-features into the optimization module, so that data diversity can also be used as a criteria for selecting base models.

Another potential source of bias is the aggregation method used to combine the outputs of the base models. If the aggregation method is biased towards certain types of predictions or certain classes, the ensemble model can inherit this bias. For example, if the aggregation method always selects the prediction of the base models with the highest confidence, it may favor overconfident predictions and be biased towards certain classes. In this regard, in the future, we will first, implement a bias monitoring module and, next, implement measures to prevent or deal with bias when it occurs.

In future work, we will expand this analyzis by including both additional datasets and larger block sizes. Indeed, we are aware that the block sizes tested are relatively small, given that the standard block size of HDFS is 128 MB. However, this analyzis is necessary to find the optimum point which, as the results point out, appears to be around 32 MB. However, although this holds true for the four datasets used, we acknowledge that this may depend on the characteristics of the datasets. Studying additional ones will allow us to be more confident in that regard.

Based on the findings of this paper, we will also implement a model management module that will track the usefulness of each base model, that is, the number of times in which it was selected to be part of an Ensemble. This module will then, over time, remove base models that are not used by H_3, freeing up resources in the cluster.

In conclusion, the work conducted allowed us to ascertain the ability of CEDEs to produce valid and useful predictive Ensembles and to determine the best block size and heuristic to create those Ensembles. Thus, it sheds light on important decisions that must be taken during its implementation and provides the grounds for those decisions.

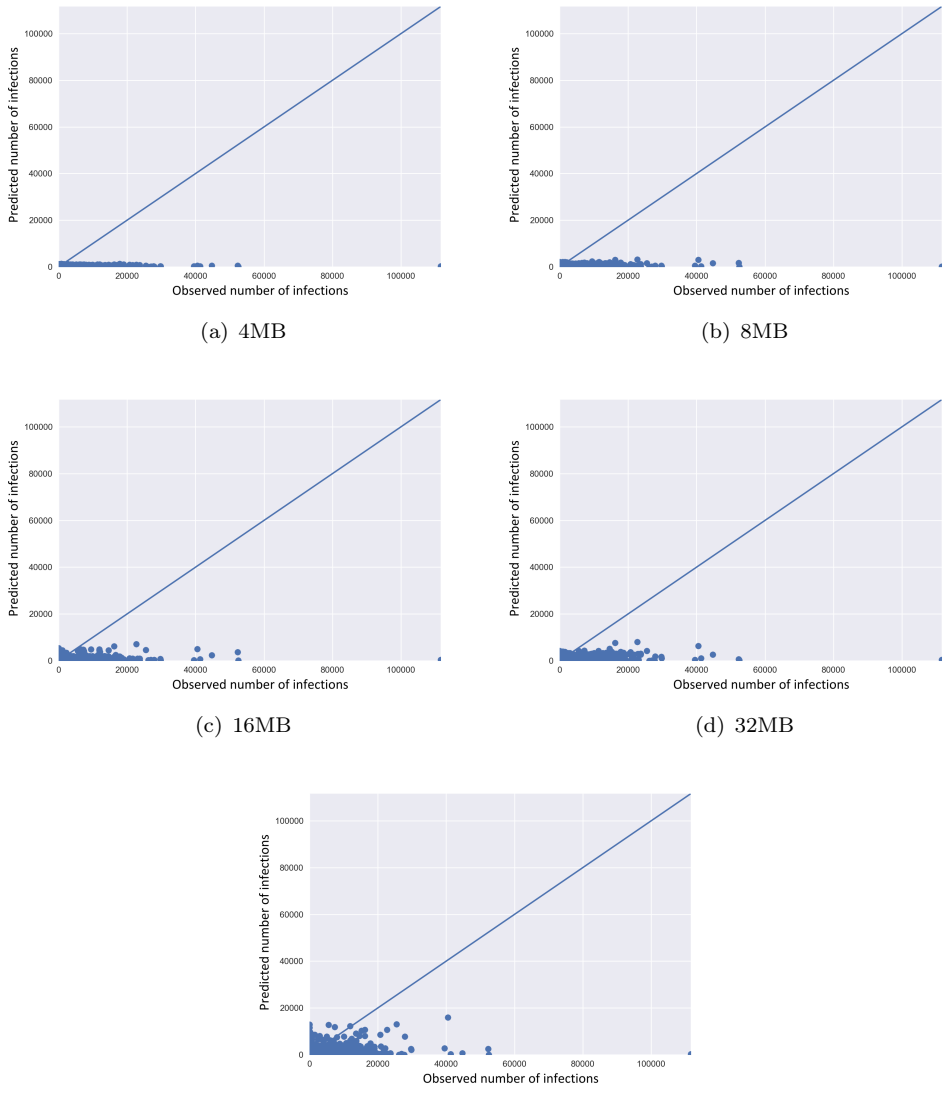
Acknowledgments

This work was supported by FCT – Fundação para a Ciência e Tecnologia within projects UIDB/04728/2020, EXPL/CCI-COM/0706/2021 and CPCA-IAC/AV/475278/2022.

References

[1] Zhou ZH. Machine learning. Springer Nature; 2021.
[2] Onan A. Bidirectional convolutional recurrent neural network architecture with group-wise enhancement mechanism for text sentiment classification. Journal of King Saud University-Computer and Information Sciences. 2022;34(5):2098–2117.
[3] Onan A. An ensemble scheme based on language function analysis and feature engineering for text genre classification. Journal of Information Science. 2018;44(1):28–47.
[4] Onan A. Sentiment analysis on product reviews based on weighted word embeddings and deep neural networks. Concurrency and Computation: Practice and Experience. 2021; 33(23):e5909.
[5] Zhou L, Pan S, Wang J, et al. Machine learning on big data: Opportunities and challenges. Neurocomputing. 2017;237:350–361.
[6] Mahesh B. Machine learning algorithms-a review. International Journal of Science and Research (IJSR)[Internet]. 2020;9:381–386.
[7] Dubey R, Gunasekaran A, Childe SJ, et al. Big data and predictive analytics and manufacturing performance: integrating institutional theory, resource-based view and big data culture. British Journal of Management. 2019;30(2):341–361.
[8] Mohammadi M, Al-Fuqaha A, Sorour S, et al. Deep learning for iot big data and streaming analytics: A survey. IEEE Communications Surveys & Tutorials. 2018;20(4):2923–2960.
[9] Gomes HM, Read J, Bifet A, et al. Machine learning for streaming data: state of the art, challenges, and opportunities. ACM SIGKDD Explorations Newsletter. 2019;21(2):6–22.
[10] Xu C, Du X, Fan X, et al. Cloud-based storage and computing for remote sensing big data: a technical review. International Journal of Digital Earth. 2022;15(1):1417–1445.
[11] Drosou M, Jagadish HV, Pitoura E, et al. Diversity in big data: A review. Big data. 2017; 5(2):73–84.
[12] De S, Panjwani M. A comparative study on distributed file systems. In: Modern approaches in machine learning and cognitive science: A walkthrough: Latest trends in ai, volume 2. Springer; 2021. p. 43–51.
[13] Andrews GR. Foundations of multithreaded, parallel, and distributed programming. Addison-Wesley; 2020.
[14] Verbraeken J, Wolting M, Katzy J, et al. A survey on distributed machine learning. Acm computing surveys (csur). 2020;53(2):1–33.
[15] Onan A, Korukoğlu S, Bulut H. Ensemble of keyword extraction methods and classifiers in text classification. Expert Systems with Applications. 2016;57:232–247.

[16] Schelter S, Biessmann F, Januschowski T, et al. On challenges in machine learning model management. 2015;.
[17] Pauloski JG, Zhang Z, Huang L, et al. Convolutional neural network training with distributed k-fac. In: SC20: International Conference for High Performance Computing, Networking, Storage and Analysis; IEEE; 2020. p. 1–12.
[18] Dong X, Yu Z, Cao W, et al. A survey on ensemble learning. Frontiers of Computer Science. 2020;14:241–258.
[19] Pham H, Olafsson S. Bagged ensembles with tunable parameters. Computational Intelligence. 2019;35(1):184–203.
[20] Roelofs R, Shankar V, Recht B, et al. A meta-analysis of overfitting in machine learning. Advances in Neural Information Processing Systems. 2019;32.
[21] Rajagopal S, Kundapur PP, Hareesha KS. A stacking ensemble for network intrusion detection using heterogeneous datasets. Security and Communication Networks. 2020; 2020:1–9.
[22] González S, García S, Del Ser J, et al. A practical tutorial on bagging and boosting based ensembles for machine learning: Algorithms, software tools, performance study, practical perspectives and opportunities. Information Fusion. 2020;64:205–237.
[23] Liu J, Huang J, Zhou Y, et al. From distributed machine learning to federated learning: A survey. Knowledge and Information Systems. 2022;64(4):885–917.
[24] Seeger M, Ultra-Large-Sites S. Key-value stores: a practical overview. Computer Science and Media, Stuttgart. 2009;.
[25] Shvachko K, Kuang H, Radia S, et al. The hadoop distributed file system. In: 2010 IEEE 26th symposium on mass storage systems and technologies (MSST); Ieee; 2010. p. 1–10.
[26] Attiya H. Concurrency and the principle of data locality. IEEE Distributed Systems Online. 2007;8(9):3–3.
[27] Botchkarev A. A new typology design of performance metrics to measure errors in machine learning regression algorithms. Interdisciplinary Journal of Information, Knowledge, and Management. 2019;14:045–076.
[28] Monteiro J, Oliveira ó, Carneiro D. Task scheduling with makespan minimization for distributed machine learning ensembles. In: 2022 IEEE 4th Eurasia Conference on IOT, Communication and Engineering (ECICE); IEEE; 2022. p. 435–438.
[29] Carneiro D, Guimarães M, Silva F, et al. A predictive and user-centric approach to machine learning in data streaming scenarios. Neurocomputing. 2021;.
[30] Carneiro D, Guimarães M, Carvalho M, et al. Using meta-learning to predict performance metrics in machine learning problems. Expert Systems. 2021;:e12900.
[31] Ramos D, Carneiro D, Novais P. Using evolving ensembles to deal with concept drift in streaming scenarios. In: proceedings of the 14th International Symposium on Intelligent Distributed Computing (IDC 2021); (Studies in Computational Intelligence; Vol. 1026). Springer; 2022.
[32] Thieu NV. Permetrics: A framework of performance metrics for artificial intelligence models ; 2020. Available from: https://doi.org/10.5281/zenodo.3951205.
[33] Tillman RE. Structure learning with independent non-identically distributed data. In: Proceedings of the 26th Annual International Conference on Machine Learning; 2009. p. 1041–1048.
[34] Angelov PP, Soares EA, Jiang R, et al. Explainable artificial intelligence: an analytical review. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery. 2021; 11(5):e1424.
[35] Ribeiro MT, Singh S, Guestrin C. " why should i trust you?" explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining; 2016. p. 1135–1144.
[36] Sundararajan M, Najmi A. The many shapley values for model explanation. In: International conference on machine learning; PMLR; 2020. p. 9269–9278.



(a) 4MB

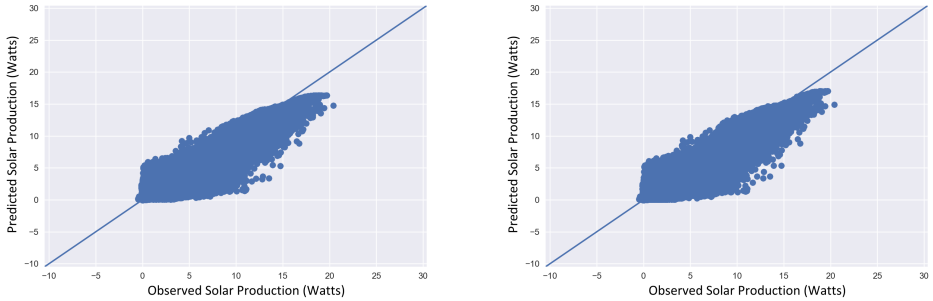
(b) 8MB

(c) 16MB

(d) 32MB

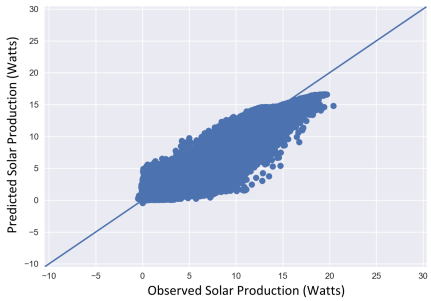
(e) 64MB

Figure 5. Predicted values vs. observed values (number of infections) for the Covid infections dataset, using different block sizes. The solid line represents the region of a perfect prediction. Base models for this time-series problem have generally poor performance since the algorithms used are not specific for time-series.

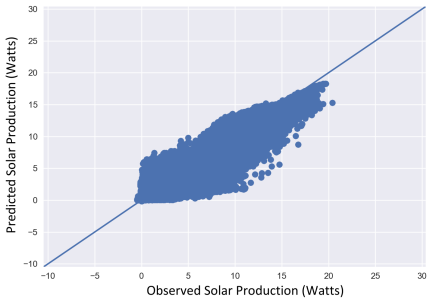


(a) 4MB

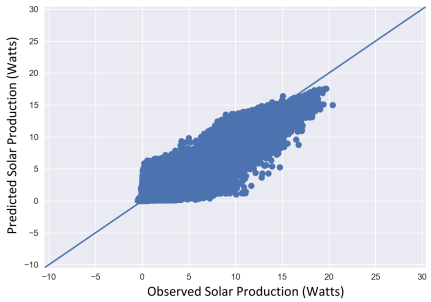
(b) 8MB



(c) 16MB

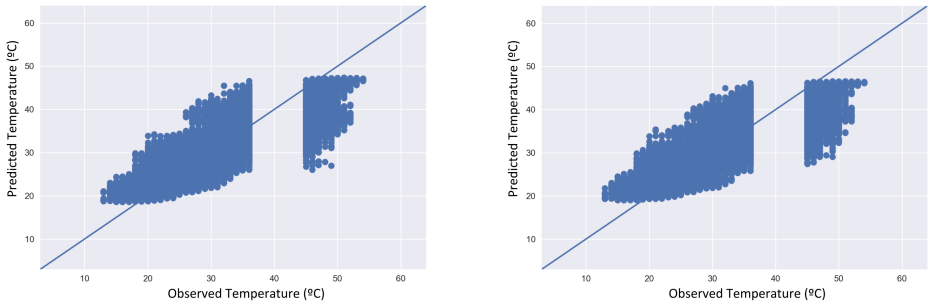


(d) 32MB



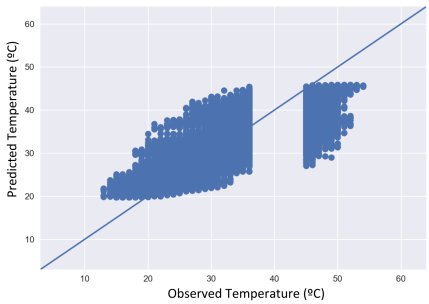
(e) 64MB

Figure 6. Predicted values vs. observed values for the Solar production dataset (Watts), using different block sizes. The solid line represents the region of a perfect prediction. The lowest error is obtained at 32MB.

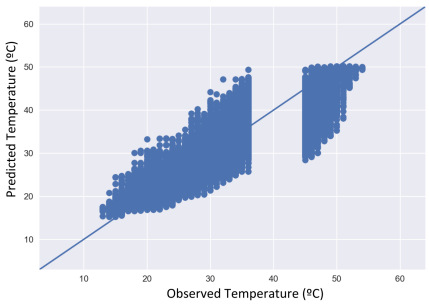


(a) 4MB

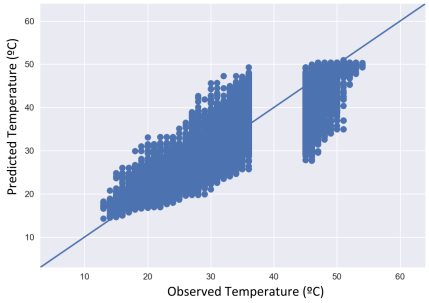
(b) 8MB



(c) 16MB

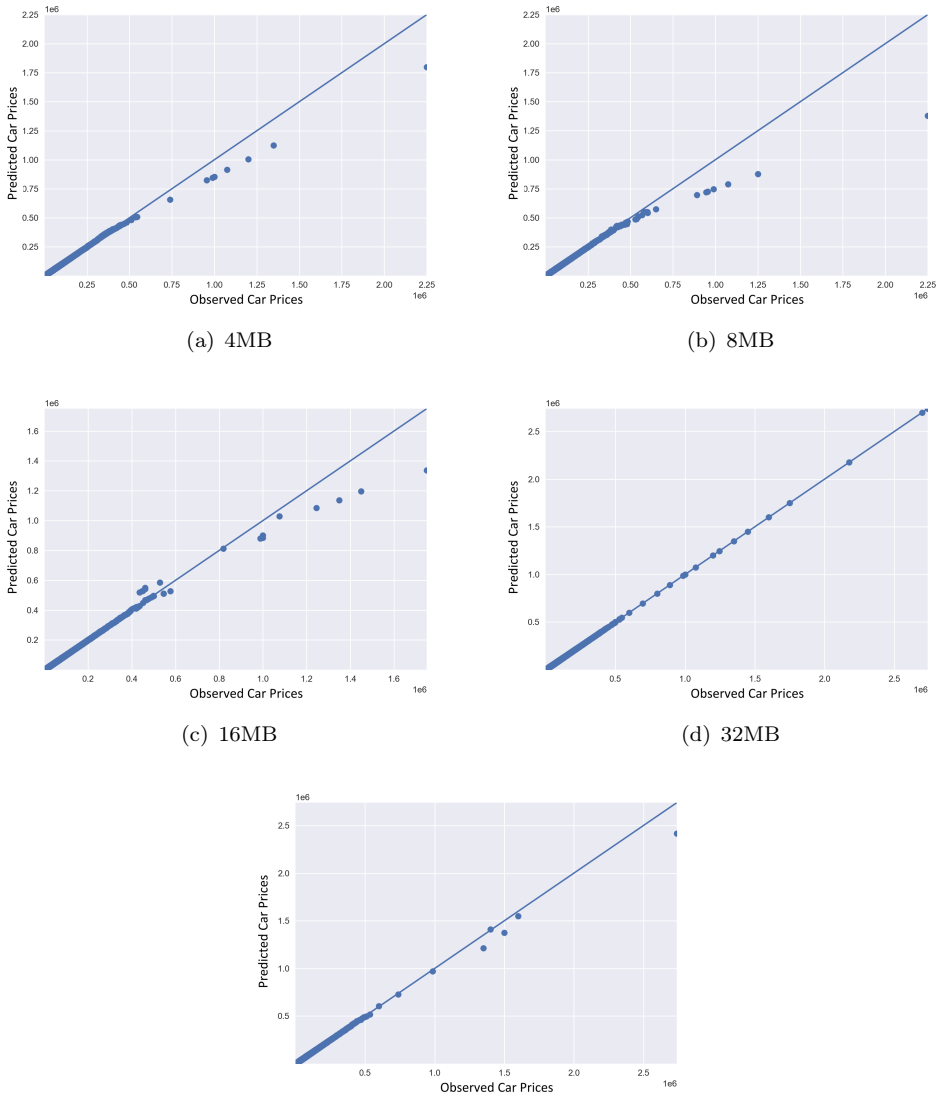


(d) 32MB



(e) 64MB

Figure 7. Predicted values vs. observed values for the Temperature dataset (°C), using different block sizes. The solid line represents the region of a perfect prediction. The lowest error is obtained at 32MB.



(a) 4MB

(b) 8MB

(c) 16MB

(d) 32MB

(e) 64MB

Figure 8. Predicted values vs. observed values for the Car prices dataset, using different block sizes. The solid line represents the region of a perfect prediction. The lowest error is obtained at 32MB.