Dynamic Management of Distributed Machine Learning Projects

Filipe Oliveira and André Alves and Hugo Moço and José Monteiro and Óscar Oliveira and Davide Carneiro and Paulo Novais

Abstract Given the new requirements of Machine Learning problems in the last years, especially in what concerns the volume, diversity and speed of data, new approaches are needed to deal with the associated challenges. In this paper we describe CEDEs - a distributed learning system that runs on top of an Hadoop cluster and takes advantage of blocks, replication and balancing. CEDEs trains models in a distributed manner following the principle of data locality, and is able to change parts of the model through an optimization module, thus allowing a model to evolve over time as the data changes. This paper describes its generic architecture, details the implementation of the first modules, and provides a first validation.

1 Introduction

Despite all the recent advances in the field of Machine Learning (ML) and related supporting areas, many new challenges keep emerging [1]. These stem mostly from the volume and diversity of data in current ML problems, as well as from the need to deliver services in real-time, which led to the emergence of streaming data, streaming analytics [2] and streaming ML [3].

ML applications nowadays require such a volume that datasets must be distributed across clusters. Consequently, ML algorithms need to learn in a distributed manner, from multiple sources of data. Moreover, these data change over time, leading to the need for models to be updated or fully retrained, which has an increasingly significant cost on the organizations' infrastructure.

Filipe Oliveira, André Alves, Hugo Moço, José Monteiro, Óscar Oliveira, Davide Carneiro
CIICESI,ESTG, Politécnico do Porto, Portugal, e-mail: {fvol,afba,hmsm,jmgm,oao,dcarneiro}@estg.ipp.pt

Paulo Novais
Algoritmi Center/Department of Informatics, University of Minho, Portugal e-mail: pjon@di.uminho.pt

In this paper we describe the initial work being developed in the context of the CEDEs project - Continuously Evolving Distributed Ensembles. CEDEs aims to build an environment for the distributed training of ML models, in which the models can evolve over time as data change, in a cost-effective way. Therefore, not only does it address the issue of learning from large datasets, but also that of learning continuously from streaming data.

CEDEs takes advantage of existing block-based distributed file systems, such as the Hadoop Distributed File System (HDFS) [4], to parallelize and distribute learning tasks, following the principle of data locality. That is, the computation is moved to where the data reside, rather than the other way around [5]. Moreover, instead of more traditional models, CEDEs uses Ensembles [6]: a base model is trained for a block of a dataset, where the data resides. Then, Ensembles are built in real time by combining available models according to criteria such as their performance or the state of the nodes where the base models reside. This is done by the optimization module that adapts the Ensemble as data changes.

The system also makes use of two other mechanisms: replication and balancing. Replication allows to automatically create replicas of blocks so that the same block is available in multiple locations simultaneously. This increases the use of storage space, but makes it easier for the optimization module to find suitable nodes for carrying out tasks. Balancing, on the other hand, distributes the blocks evenly across the cluster, so that the load when executing tasks is appropriately distributed.

Finally, CEDEs also stores the base models themselves in the HDFS, which are then replicated. This means that, not only, the Ensemble itself is distributed, which speeds up predictions, but also that, thank to replication, multiple nodes will have the same base models available for making predictions. Once again, it is up to the optimization module to decide which base models to use and in which nodes to make predictions. This paper describes the implementation, operation and testing/validation of the core modules of this system.

2 Conceptualization

This section details the proposed system in terms of its main components and functionalities. It first describes the underlying data model (Section 2.1), and then the proposed architecture (Section 2.2).

2.1 Data Model

The data model that supports an instance of the proposed system is composed of seven main entities, as depicted in Figure 1, which are described next.

Each instance of the system is expected to address a specific Machine Learning problem or domain (e.g. Healthcare, Fraud Detection, Finance). While the system

allows multiple organizations to be part of an instance, they are expected to be in some way related to the domain of that instance. Moreover, organizations can perform different roles, depending on how they use the available services. For example, an organization may primarily be a contributor of data, when it contributes with data and, consequently, models, but makes little to no use of predictive services. Alternatively, an organization may essentially be a consumer, when it contributes with little to no data, but makes a heavy use of predictive services. Any intermediary range is also possible. Given that this characterization is volatile through time (i.e. the stance of each organization may change), it is not part of the data model. Instead, this is derived from the blockchain, as detailed further below.

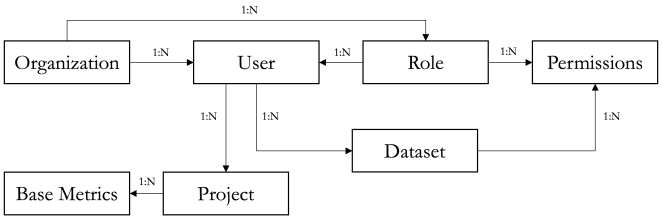

Fig. 1 Simplified view of the Data Model that supports the proposed system.

Each organization may have multiple users, who are the central entity in the data model. A user may create and edit ML projects and import or access datasets. The access of each user to each dataset is governed by a system of roles and permissions. Each organization may define multiple roles, according to its internal organization (e.g. Data Scientist, ML Engineer, Data Engineer). Each user may then have one or more roles inside her/his organization. Permissions are then attributed on the basis of user roles. Moreover, when a user uploads a new dataset, she/he becomes the owner of the dataset. This is important to determine the provenance of the data.

When a user creates a new ML project she/he must provide some information, including: name and general description of the project, the underlying dataset, which algorithm(s) is/are to be used in the training of the base models, configurations of the algorithms, etc.

Multiple algorithms can be used in a single project. This is done when the user wants to build a heterogeneous ensemble, composed of models of different types. In this case, the user must provide the relative proportion of each algorithm (e.g. 50/30/20) as well as the configuration to use for each algorithm. The configuration is dependent 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). This allows for a great range of different ensembles to be built.

When the user requests for a given model to be trained, in the context of a specific ML project, this actually results in the training of multiple base models: one for each block (excluding block replicas) of the dataset. While this is completely transparent

to the user, the resulting information is stored in the Base Metrics entity. This table has one line for each base model of each ensemble and contains, among others, data describing the type of algorithm used, the node where the training took place, the resulting performance metrics (e.g. RMSE, MAE, R^2) and the computational cost of training the model (e.g. training time, memory and CPU consumption).

There are also relevant additional data, that are not stored in this data model. Namely, we do not store the meta-data regarding the blocks of each dataset and their location, as these are dynamic and change through time. This information is requested in real-time from the HDFS, when needed, lest we risk using outdated information. The same goes for the location of the base models of a project when there's the need to constitute an ensemble and making predictions.

The architecture of the proposed system, which makes use of this data model, is described in the following Section.

2.2 Architecture

The architecture of the proposed solution is composed of a number of major modules, which are described in this section. The central element is the Storage Layer. This is implemented in the form of an HDFS cluster. Here, large datasets are split into smaller chunks called blocks, of a fixed size (typically 128MB). Multiple data sources can be considered, ranging from files in different formats (e.g. CSV, Parquet, Avro) to databases. The whole process starts with a user uploading a data source into the HDFS, which is then split, distributed and replicated across the cluster.

Once this process finishes, the dataset becomes available to be used in a new ML project. The user can thus create a new project, select the intended dataset, and provide the necessary information already detailed in the previous Section. Once the configuration of the project is complete, the user may start the training of a new ensemble. This process is managed by the Coordination module, which interacts with the Optimization and Metadata modules.

The training of an ensemble takes place in a distributed manner and takes advantage of two aspects: the fact that the blocks that constitute each file are replicated and available in multiple nodes simultaneously, and the principle of data locality. An ensemble is thus trained by training one base model for each block of a file (excluding replicas) which, when combined, constitute the ensemble. There is thus the need to select the most appropriate node where to train each base model for each block, according to criteria such as the state of each node of the cluster where that block is available.

Specifically, the Coordination module needs the following inputs:

Block location. For a given ML problem, the Coordination module needs to know what blocks are part of the respective dataset, and where each of its replicas are available at the moment of training. Each block is typically available in at least 3 nodes, but this is dependent on the configuration of the HDFS. The number of blocks of a dataset determines the number of base models to be trained (1:1 relationship).

This information is obtained from the Metadata module;

Task cost prediction. The duration and cost, in terms of computational resources, of the training of a given base model depends on multiple factors, including the size of the dataset (lines and columns) and the type and configuration of the algorithm used. To have an estimate of the cost of each individual task (i.e. the training of each base model), an approach based on meta-learning was developed that uses multiple meta-models, one to predict each intended cost metric, based on metadata collected from a large group of ML problems. This approach, which provides a fairly good estimate of task complexity/cost is further described in [7, 8]. This information is obtained from the Optimization module;

Nodes state. Another relevant aspect to consider when deciding where to train base models is the state of each node in the cluster. The state of a node is characterized by its health (which is calculated and maintained by Hadoop's health checker service), its current load (e.g. memory, CPU, disk) and the size and cost of its tasks queue. This information is provided by the Optimization module;

Task allocation. In order to implement the actual training of the models of the ensemble, the Coordination module also needs information regarding where each task (base model training) will take place. For instance, when there are three candidate nodes to train a base model for a specific block (thanks to replication), the system will have to choose the best candidate node. The Optimization module carries out a global optimization heuristic, which will provide as output the best allocation of all the necessary tasks across the cluster, based on the elements mentioned in the previous two points (i.e. Tasks cost prediction and nodes state);

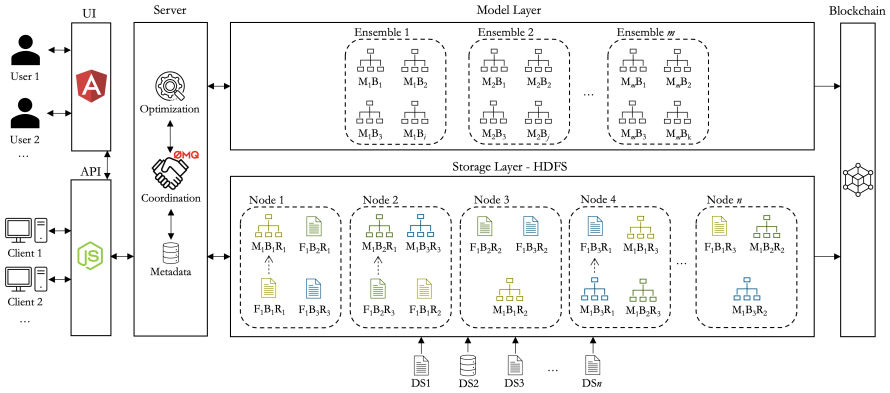

Fig. 2 Depiction of the proposed architecture with a possible sample scenario.

Once the Coordination module receives the tasks allocation for a given ML project, it is ready to start the training process. This process is implemented in ZeroMQ: a brokerless asynchronous messaging library that implements multiple socket communication patterns, useful for implementing distributed systems.

As the distributed training process goes on, each worker node updates the coordinator at regular intervals about its individual progress. This information can thus be accessed in real time by the client applications. As the training of each individual base model ends, the coordinator is notified and the worker node can move on to the next task in its queue, if there is one.

As soon as the training of a base model ends, another process takes place. The model is serialized and is stored in the HDFS. Automatically, the base model will be replicated and distributed across the cluster, according to the replication and balancing factors. This means that soon after the training of a base model, it will be available for making predictions in multiple nodes of the cluster.

This is illustrated in Figure 2, in which the following nomenclature is used: $F_i B_j R_k$ represents a replica k of block j of file i. The same logic is used for models, which start with the letter M. An arrow between a specific block replica and a model means that that base model was trained from that block, on that node. Replicas of that base model may however exist in other nodes, created through the replication mechanism of Hadoop after the base model was stored in the HDFS.

Thus, when the client requests for predictions from a given ensemble, a similar process takes place to that of training models. First, the Coordination module needs information regarding which of the base models will be used as part of the ensemble. Indeed, depending on the configuration of the ML project, not all the base models may be part of the ensemble. The user may decide to have an ensemble with a number of base models that is smaller than the number of blocks of the dataset. In that case, the Optimization module will decide, according to the performance metrics of the base models and the state of the nodes where they reside, the best group of base models to constitute the ensemble. In the past, we have implemented this process using genetic algorithms, as described in [9]. However, a more efficient approach is now being implemented.

Once the Coordination module receives information about task allocation, it distributes the tasks accordingly by the cluster by means of messages, and awaits for the predictions of each base model. The predictions are then combined to calculate the final prediction by the ensemble. To this end, the mean is used for regression problems, and the mode is used for classification.

Ensembles are thus abstract constructs, depicted in Figure 2 in the Model Layer, built out of a selection of available base models for a given ML problem.

There are three mode modules in this architecture. The Blockchain module is used to record all the actions of users or clients in the system, including who imported datasets, who requested the training of models, who requested predictions etc. The API module includes all the endpoints necessary for interacting with the services of the system, which in turn communicate with the Coordination module, when applicable, through ZeroMQ messages. This API can be used by external client applications. Finally, there is also a User Interface meant to be used by Human users, which communicates with the previously mentioned API.

The described system is not fully implemented yet. Work is currently undergoing in the Blockchain and Optimization modules. Currently, the Coordinator assigns tasks randomly among available and valid candidate nodes, for each ML project.

3 Validation

This section describes the methodology followed for validating the proposed system, and its results. Given the lack of a physical infrastructure, the distributed system was simulated as a Docker application with 4 containers: 1 acting as the coordinator and 3 acting as workers. Each container is based on the same image, built on top of the ubuntu:bionic image, and in which all the Hadoop ecosystem was installed. To simulate the actual conditions of a real cluster, Hadoop was installed in distributed mode.

The characteristics of the datasets used for testing the system are described in Table 1. Given that the sizer of the datasets is relatively small for what is standard in an Hadoop cluster, HDFS's block size was reduced to 8MB, to force a larger number of blocks and parallelism.

dataset	rows	columns	size	blocks
city temperatures	2906327	8	140.6 MB	18
mnist	70000	785	127.9 MB	16
sales records	1048575	14	130.9 MB	16

Table 1 Brief characterization of the datasets used for validation.

For each dataset, an Ensemble was trained using the proposed system. A heterogeneous Ensemble was used, containing a mixture of three algorithms from the scikit-learn library: random forest (sklearn.ensemble.RandomForestClassifier), decision trees (sklearn.tree. DecisionTreeClassifier) and neural networks (sklearn. neural_network.MLPClassifier). The configuration of algorithms used is detailed in Table 2. String features were transformed using scikit-learn's label encoder.

algorithm	parameter	value
decision tree	max_depth	5
	min_samples_split	2
	max_leaf_nodes	5
	ccp_alpha	0
neural network	hidden_layers_size	[5,2]
	activation	relu
	solver	adam
	alpha	0.0001
	learning_rate	constant
	max_iter	200
random forest	nr_estimators	5

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

In the proposed system, the user can also select the weight of each algorithm in the Ensemble (Figure 3), which results in a proportional number of models for each algorithm in the Ensemble. For instance, given a dataset split into 18 blocks and the weights of 50, 30 and 20 for the algorithms random forest, decision trees and neural networks, respectively, the Ensemble would have 12 random forest, 3 neural networks and 3 decision trees. For the purpose of these tests, these were the weights used. Moreover, the holdout method was used to evaluate each base model, with 75% of the data used for training and the remaining 25% used for testing. The performance of the model is obtained by computing the average value of a given performance metric (e.g. RMSE, MAE).

In order to compare the results obtained with a baseline, we trained 3 random forests using a commercially available tool (H2O), using the pre-defined configuration of the algorithm, namely 50 trees with a maximum depth of 20. The results are compared with those of the proposed system in Table 3 in terms of the RMSE, R^2 and training time. The performance is equivalent, both in terms of predictive performance or training time. However, it must be stated that the goal at the time is not to have the most accurate models possible but rather to test if all the components of the system are working as intended.

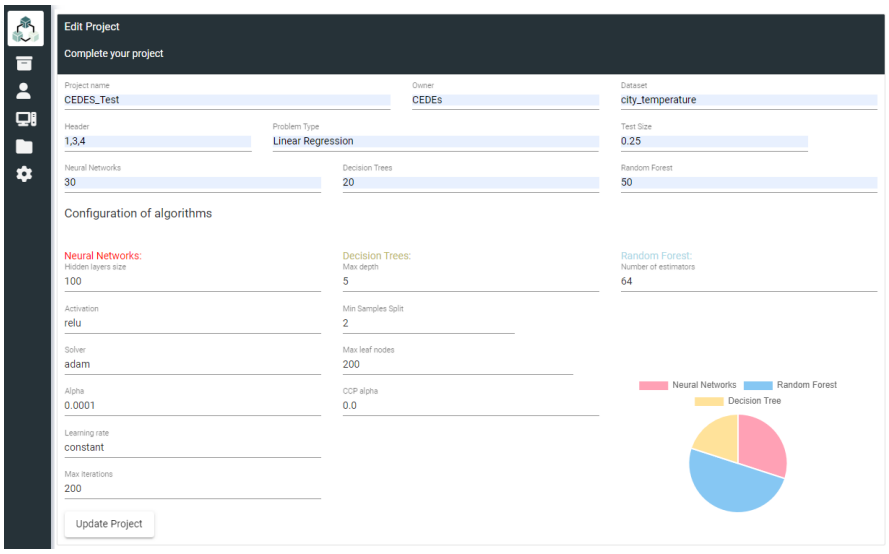

Fig. 3 Prototype of the UI: page for editing the properties of a new ML project.

	proposed approach			commercial tool		
dataset	rmse	r2	time (s)	rmse	r2	time (s)
city temperatures	21.40	0.51	120.06	20.13	0.61	128.44
mnist	1.82	0.59	244.50	0.86	0.91	101.23
sales records	63198.12	0.93	17.34	7623.67	0.99	577.5

Table 3 Results of the models trained using the proposed system.

4 Discussion, Conclusions and Future Work

This paper described the initial steps in the implementation of a distributed learning system that has some key innovative features: it trains models for big distributed datasets following the principle of data locality; it decides in which nodes to carry out the training and predicting tasks according to the state of the cluster; it allows to use heterogeneous Ensembles; and it abstracts all the complexity of distribute learning and associated coordinated tasks from the user. The optimization mechanism is especially important as it will allow for the performance of the ensemble to be improved by adjusting which base models make part of it and with which weight.

The main goal of this paper was to validate the proposed system and test its functionalities rather than to obtain the best possible models. For this reason, we did not invest time in finding the best possible configuration for each ML problem and used the same configuration for all. This may explain the relatively poorer performance of the trained models when compared to those of a commercial tool. Figure 4 shows how the value of the RMSE varies significantly according depending on the base model for the "city temperatures" ensemble, hinting that improvements could be achieved by tweaking the configurations used.

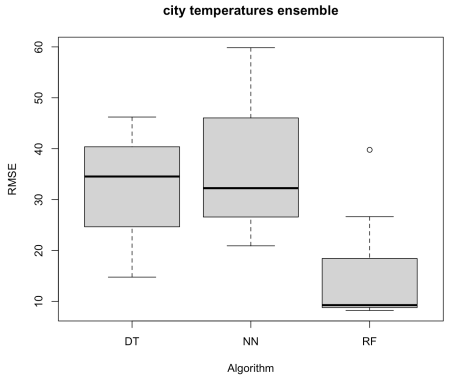

Fig. 4 Distribution of the RMSE for the 18 base models trained for the "city temperatures" dataset, by algorithm.

Moreover, given that we have no dedicated infrastructure yet, we have implemented and tested the system on a Docker instance. In this scenario, we chose to

deliberately use a very small block size for HDFS (8MB), in order to increase the number of blocks, so as to validate the communication protocol and overall performance of the system. This leads to very small training sets, which may lead to relatively poorer models, and in turn, lead to generally poor ensembles.

Work now continues on the implementation of the Optimization and Blockchain modules. Moreover, a streaming module based on Kafka is now also being implemented, that will allow data to change over time and new base models to be added to the Ensemble. The Optimization module will be used to determine which models will make part of a given Ensemble in real time, allowing for the Ensemble to evolve over time, as new models are added and others removed. This is only possible due to the decisions taken during the implementation of this system, that allow for this flexibility that is absent in other similar tools. Ultimately, it will allow to better deal with the significant challenges that the field of ML faces today, namely in what concerns learning continuously from large volumes of streaming data.

Acknowledgments

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

References

1. Zhou, L., Pan, S., Wang, J., Vasilakos, A.V.: Machine learning on big data: Opportunities and challenges. Neurocomputing **237** (2017) 350–361
2. Mohammadi, M., Al-Fuqaha, A., Sorour, S., Guizani, M.: Deep learning for iot big data and streaming analytics: A survey. IEEE Communications Surveys & Tutorials **20**(4) (2018) 2923–2960
3. Gomes, H.M., Read, J., Bifet, A., Barddal, J.P., Gama, J.: Machine learning for streaming data: state of the art, challenges, and opportunities. ACM SIGKDD Explorations Newsletter **21**(2) (2019) 6–22
4. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The hadoop distributed file system. In: 2010 IEEE 26th symposium on mass storage systems and technologies (MSST), Ieee (2010) 1–10
5. Attiya, H.: Concurrency and the principle of data locality. IEEE Distributed Systems Online **8**(9) (2007) 3–3
6. Dong, X., Yu, Z., Cao, W., Shi, Y., Ma, Q.: A survey on ensemble learning. Frontiers of Computer Science **14**(2) (2020) 241–258
7. Carneiro, D., Guimarães, M., Silva, F., Novais, P.: A predictive and user-centric approach to machine learning in data streaming scenarios. Neurocomputing (2021)
8. Carneiro, D., Guimarães, M., Carvalho, M., Novais, P.: Using meta-learning to predict performance metrics in machine learning problems. Expert Systems (2021) e12900
9. Ramos, Diogo, C.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). Volume 1026 of Studies in Computational Intelligence., Springer (2022)