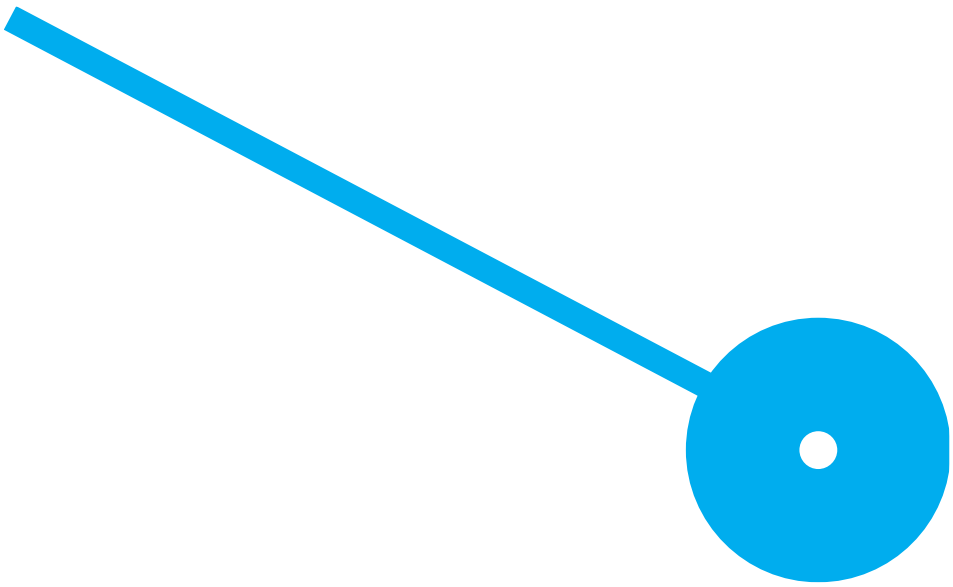
ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO
POLITÉCNICO
DO PORTO

MESTRADO
ENGENHARIA INFORMÁTICA

Dynamic Management of Distributed Machine Learning Problems

Filipe Vamonde Oliveira

11/2023



ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO
POLITÉCNICO
DO PORTO

MESTRADO
ENGENHARIA INFORMÁTICA

Dynamic Management of Distributed Machine Learning Problems

Filipe Vamonde Oliveira
8170164

Orientador

PhD Davide Rua Carneiro

Dissertação apresentada para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Informática pela Escola Superior de Tecnologia e Gestão do Instituto Politécnico do Porto.

11/2023

Declaração de Integridade

Eu, Filipe Vamonde Oliveira, estudante nº 8170164, do Mestrado de Engenharia Informática da Escola Superior de Tecnologia e Gestão do Instituto Politécnico do Porto, declaro que não fiz plágio nem auto-plágio, pelo que o trabalho intitulado "Dynamic Management of Distributed Machine Learning Problems" é original e da minha autoria, não tendo sido usado previamente para qualquer outro fim. Mais declaro que todas as fontes usadas estão citadas, no texto e na bibliografia final, segundo as regras de referenciação adotadas na instituição.

Agradecimentos

Em qualquer projeto existe um conjunto de pessoas que o tornam possível, e sem elas, este projeto não estaria concluído.

Agradeço em primeiro lugar ao meu orientador, professor Dr. Davide Carneiro, pela maneira como me acolheu como seu discípulo. Agradeço pela confiança depositada, pelas reuniões que se disponibilizou, correções e sugestões, assim como pela revisão e orientação do projeto.

Agradeço por tudo o que aprendi. E principalmente, por tudo que tive oportunidade de fazer.

Aos meus colegas, pelo suporte, motivação e companhia.

Obrigado à Escola Superior de Tecnologia e Gestão por me ter acolhido no meu percurso profissional, a todo o corpo docente, e pela disponibilidade destes.

Um agradecimento especial também para quem não contribuiu de forma direta para o sucesso deste projeto, mas ainda assim, sem eles também não seria possível, incluindo a família.

"Distributed machine learning allows us to break down large, complex problems into smaller, more manageable parts, enabling us to tackle them with greater efficiency and accuracy."

FEI-FEI LI

RESUMO

Machine Learning (ML) e Inteligência Artificial (IA) são dois termos intimamente relacionados. A Inteligência Artificial é uma disciplina que busca criar máquinas que tenham a capacidade de imitar as habilidades cognitivas humanas, como aprendizagem, raciocínio, perceção, e tomada de decisão. *Machine Learning* é uma das técnicas de IA que permite às máquinas aprenderem a partir de dados sem serem explicitamente programadas.

O crescimento exponencial dos dados nas últimas décadas tem sido um dos principais fatores impulsionadores do avanço da Inteligência Artificial e de *Machine Learning*. As empresas e organizações recolhem dados em volumes cada vez maiores, incluindo informações de transações financeiras, registos médicos, dados de sensores IoT e muito mais. Esses dados são cruciais para impulsionar a inovação e o progresso, mas podem ser muito complexos e difíceis de serem analisados manualmente.

É aqui que entra o *Machine Learning*, que permite que as máquinas aprendam e automatizem a análise de grandes conjuntos de dados. Essa abordagem reduz o tempo e o esforço necessários para realizar análises complexas, além de fornecer *insights* valiosos que podem ser usados para melhorar as operações de negócios, aumentar a eficiência e tomar decisões mais informadas.

À medida que os dados continuam a crescer em tamanho e complexidade, é necessário adotar novas abordagens e sistemas para lidar com eles de forma eficiente. Uma das maneiras pelas quais isso está a ser feito é através do desenvolvimento de técnicas *Machine Learning* mais avançadas, como redes neuronais profundas e algoritmos de aprendizagem por reforço (*Reinforcement Learning*) (RL), que podem lidar com conjuntos de dados maiores e mais complexos de forma mais eficaz. Além disso, o uso de tecnologias como a computação em cloud e o processamento de dados distribuídos também pode ajudar a reduzir o consumo de recursos computacionais e a tornar a análise de dados mais escalável.

Desta forma, a solução proposta, surge para colmatar alguns dos desafios que surgiram com o aumento do volume dos dados. Um sistema de *Machine Learning* distribuído que corre sobre um *cluster Hadoop* e tira vantagem das funcionalidades de replicação, balanceamento e distribuição por blocos. Permite o treino de modelos de uma forma distribuída seguindo o princípio de *data locality*, sendo capaz de alterar partes do modelo através de um módulo de otimização, portanto possibilitando a evolução do modelo ao longo do tempo consoante novos dados chegam.

Palavras-chave: *Distributed Machine Learning. Distributed File System. Hadoop. Machine Learning*.

ABSTRACT

Machine Learning (ML) and Artificial Intelligence (AI) are two closely related terms. Artificial Intelligence is a discipline that seeks to create machines that have the ability to mimic human cognitive skills, such as learning, reasoning, perception, and decision making. Machine Learning is one of the AI techniques that allows machines to learn from data without being explicitly programmed.

The exponential growth of data in recent decades has been one of the main driving factors of AI and Machine Learning advancement. Companies and organizations collect data in increasingly large volumes, including financial transaction information, medical records, IoT sensor data, and more. This data is crucial for driving innovation and progress, but can be too complex and difficult to analyze manually.

This is where Machine Learning comes in, allowing machines to learn and automate the analysis of large data sets. This approach reduces the time and effort required to perform complex analyses, as well as providing valuable insights that can be used to improve business operations, increase efficiency, and make more informed decisions.

As data continues to grow in size and complexity, new approaches and systems are needed to handle it efficiently. One way this is being done is through the development of more advanced Machine Learning techniques, such as deep neural networks and reinforcement learning algorithms, which can more effectively handle larger and more complex data sets. In addition, the use of technologies such as cloud computing and distributed data processing can also help reduce the consumption of computational resources and make data analysis more scalable.

Thus, the proposed solution arises to address some of the challenges that have emerged with the increase in data volume. A distributed machine learning system that runs on a Hadoop cluster and takes advantage of replication, balancing, and block distribution capabilities. It allows models to be trained in a distributed manner following the principle of data locality, being able to change parts of the model through an optimization module, thus enabling the model to evolve over time as new data arrives.

Keywords: Distributed Machine Learning, Distributed File System, Hadoop, Machine Learning.

Contents

List of Figures

List of Tables

List of Acronyms

AI Artificial Intelligence

CEDES Continuously Evolving Distributed Ensembles

ANN Artificial Neural Network

API Application Programming Interface

CD Continuous Delivery

CI Continuous Integration

CSV Comma-separated values

DRF Distributed Random Forest

DT Decision Tree

ESTG Escola Superior de Tecnologia e Gestão do IPP

EU European Union

HTTP Hypertext Transfer Protocol

IQR Interquartile range

JSON JavaScript Object Notation

MAE Mean Absolute Error

ML Machine Learning

MSE Mean Squared Error

RMSE Root Mean Squared Error

UI User Interface

Chapter 1

Introduction

The field of Machine Learning (ML) [1] has seen many advancements [2, 3, 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].

New systems resulting from research and development efforts often have a common trait - they are distributed. This means they encompass not only distributed storage but also distributed processing, specifically model training and serving. Although such systems [10] enable handling larger data volumes that cannot be accommodated in a monolithic system's memory or storage, they necessitate increased coordination and synchronization efforts. These efforts not only entail computation but also involve the complexity of the algorithms employed, which may need adaptation or fresh development.

Parallel or distributed connectionist algorithms, like neural networks or deep learning, have been devised to reduce training time with large datasets, one of their main drawbacks. In these cases, the challenge is to maintain convergence and efficiency at scale. For instance, the authors propose Kronecker-factored Approximate Curvature (K-FAC) [11] as a Fisher Information Matrix approximation [12] for use in natural gradient optimizers, validating it with compelling results.

Another approach is creating ensembles [13], which are groups of simpler models or weak learners trained with parts of the data. Although each base model is typically poor when considered individually, grouping them into an ensemble through some heuristic often yields better results than using a single complex model.

There are various types of ensembles, some of which can be implemented straightforwardly in a distributed manner. Bagging Ensembles [14] are made up of a group of base models trained with different parts of the data selected randomly. Bagging Ensembles are highly resilient to overfitting [15] and can be naturally distributed, with different base models trained on data selected from different parts of the dataset. Each base model has an equal weight in computing the final prediction in a Bagging Ensemble, regardless of its quality.

Stacking Ensembles [16], in contrast, involve training multiple models on the same data, which makes it challenging to distribute the algorithm if the data is distributed. The algorithms used to train the models must be capable of distributed learning. In Stacking Ensembles, each base model might have a different weight when making predictions, or some base models might not be considered at all, with the final stage

of training an additional model to learn how to best combine the predictions.

Boosting Ensembles [17] add base models sequentially, with each new model attempting to minimize the error on instances that were more poorly classified by the previously trained one. Boosting Ensembles are more prone to overfitting, particularly in scenarios with outliers, and are less likely to be distributed or parallelized since models are trained sequentially.

There are various approaches to distributed learning, each with its own strengths and weaknesses, and the most appropriate one can only be determined on a case-by-case basis [17]. However, when it comes to suitability for distributed learning, an ordering can be proposed, with Bagging being the most appropriate and Stacking being the least suitable.

In recent years, additional challenges have arisen in distributed learning, as data sources and storage have grown in size and are spread across different regions with varying privacy and data protection regulations. To address this, Federated Learning (FL) [18] has emerged, allowing organizations, countries, or regions to collaboratively train machine learning models in a distributed manner while respecting data ownership and privacy by keeping the data decentralized and training and sharing the models.

The CEDEs project, presented in this dissertation, integrates the concepts of Ensembles, Distributed Learning, and Federated Learning. However, CEDEs goes beyond traditional Ensembles as it enables Ensembles to evolve effortlessly over time without requiring a full re-training as data changes. Ensembles are treated as logic constructs that can be easily set up based on user requirements and the state of the cluster.

This dissertation analyzes the relationship between data block size, degree of parallelism, and the impact on the quality of the Ensemble in CEDEs. This relationship is detailed in Section 4, with the aim of finding the optimal point at the intersection of these conflicting requirements. A smaller block size increases the degree of parallelism, but also incurs a coordination overhead, while a larger block size may defeat the purpose of having an Ensemble. The impact of block size on model quality is not straightforward and will be studied in-depth.

Optimizing these factors is critical for CEDEs and other distributed learning systems to efficiently manage cluster resources. By finding the optimal point, CEDEs can achieve a more efficient evolution of Ensembles, enhancing their predictive performance over time.

1.1 Project CEDEs

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, in a cost-effective way. This addresses not only the challenge of handling large volumes of data but also the capability of learning continuously from streaming data.

CEDEs takes advantage of existing block-based distributed file system, such as Hadoop Distributed File System (HDFS) [19] to parallelize and distribute learning tasks efficiently by following the principle of data locality [20] which means that the computation is operating where data resides, instead of the other way around. Rather than conventional models, CEDEs uses Ensembles [13], where a dataset is separated by blocks and a base model is trained for each block and Ensembles are created 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 new data arrives.

1.1.1 Work operationalization

In the pursuit of conducting the project, an Agile methodology was adopted, specifically SCRUM, as the primary framework to guide the investigation. The implementation of SCRUM allowed for a dynamic and iterative approach, emphasizing adaptability and flexibility to accommodate the evolving nature of the research process. A cohesive team of dedicated researchers collaborated in order to facilitate the management and development of tasks, each member contributing unique expertise and skills to the project. Weekly meetings were an integral part of the project endeavor, providing an opportunity to assess progress, discuss challenges, and refine our research objectives. These regular gatherings facilitated effective communication and coordination, creating an environment conducive to collective problem-solving and continuous improvement. Overall, the utilization of Agile practices, complemented by a talented and committed research team and structured weekly meetings, greatly contributed to the success and efficiency of this academic pursuit. This project was funded by FCT and all the team members involved obtained a scholarship to work on it. The gitlab tool was used to define tasks and version control as the project evolved and all the code that brings this system to life is available in `https://gitlab.estg.ipp.pt/dcarneiro/cedes`.

1.2 Objectives

This project takes a part of the CEDEs project and aims to develop an environment that allows users to train Machine Learning models in a distributed manner. The goal is to create a platform that enables users to train models more efficiently and accurately, by distributing the workload across multiple machines. This distributed approach to machine learning training has the potential to significantly reduce training times and improve the accuracy of models.

In addition to developing the distributed training environment, the project aims to investigate the impact of the block size on the quality of the base models. The block size refers to the size of the data chunks that are distributed across the machines for training. By studying the impact of block size on the quality of the models, the project aims to determine the optimal block size for distributed training.

Furthermore, the project will explore different approaches for attaching weights to the predictions of each base model, using the quality of the model as a metric. This will involve testing various methods for weighting predictions, in order to determine the most effective approach for improving the performance of the ensemble.

Overall, the objectives of the project are to develop an efficient and accurate platform for distributed Machine Learning training, to investigate the impact of block size on model quality, and to explore different approaches for weighting predictions to improve model accuracy.

1.3 Research Methodology

The research methodology utilized for this project was Design Science Research (DSR). DSR is a research methodology that focuses on the creation and evaluation of innovative artifacts, such as technologies, processes, and systems. The goal of DSR is to develop practical solutions to real-world problems, while also advancing scientific knowledge in the field.

DSR, as it shows in fig. 1, follows a cyclical process that involves several stages, including problem identification, artifact design, artifact evaluation, and knowledge dissemination. In the problem identification stage, researchers identify a real-world problem that needs to be addressed. The artifact design stage involves creating a prototype of the solution, which is then evaluated in the artifact evaluation stage. The results of the evaluation are then used to refine the artifact, and the knowledge gained is disseminated in the final stage.

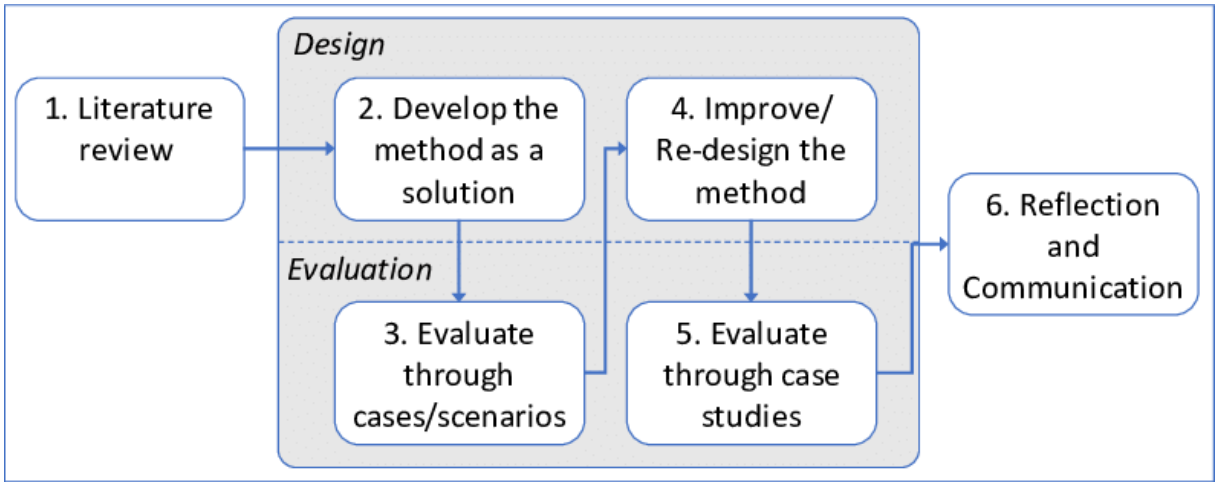


Figure 1: Visual explanation of DSR research methodology..

DSR is a highly iterative and collaborative process that involves both researchers and practitioners. By combining scientific rigor with practical application, DSR has the potential to create innovative solutions that address real-world problems while also advancing scientific knowledge in the field.

1.4 Structure

The document is structured into several chapters that provide a comprehensive overview of the project and its research methodology.

The first chapter is the introduction, where an overview of the project is presented along with the objectives and the research methodology used for its development.

The second chapter is the theoretical background, where all the related concepts within the scope of the project are discussed in detail and also existing systems that compete with the conceptualized and implemented system are analyzed and critiqued. A critical analysis is made between these systems and the one developed in this project.

The third chapter focus on the architecture of the system. In the fourth chapter, the technologies used and how they communicate with each other are discussed and also the functioning of the system is explained, along with how it uses the technologies discussed in the previous chapter to achieve its objectives.

The forth chapter focuses on the results obtained. Here, the studies that were conducted and their reasons are presented in detail, along with the outcomes and any limitations encountered during the research.

Finally, the fifth chapter presents the Scientific Dissemination, Acknowledgements and Conclusions respectively. In this chapter, is presented the articles that were written by the author as part of this project,

the acknowledgements, analysis, conclusions drawn from the study that were presented in the results chapter, and a small allusion to future work.

Overall, the document is structured in a logical and organized manner that enables the reader to gain a comprehensive understanding of the project, its research methodology, and its outcomes.

Chapter 2

Theoretical Background

2.1 Machine Learning

Machine learning, a subset of Artificial Intelligence, encompasses a diverse range of algorithms and methodologies that enable computer systems to learn from data and improve their performance over time. Two fundamental types of Machine Learning paradigms widely employed in various applications are supervised learning and unsupervised learning.

Supervised learning involves training a model on labeled data, where each data point is associated with a known outcome or target. The goal of supervised learning is to enable the model to generalize patterns from the provided labeled examples and make accurate predictions for new, unseen data. During the training process, the model is iteratively adjusted to minimize the discrepancy between its predictions and the ground truth labels. Popular algorithms in supervised learning include linear regression [21], decision trees [22], support vector machines [23], and neural networks [24]. Applications of supervised learning abound in tasks such as image and speech recognition, sentiment analysis, and medical diagnosis, where the model learns to associate input features with specific predefined outputs.

On the other hand, unsupervised learning operates on unlabeled data, meaning there are no explicit target labels provided during training. The primary objective of unsupervised learning is to extract underlying patterns, structures, or relationships from the data without any explicit guidance. Clustering [25] and dimensionality reduction [26] are common techniques employed in unsupervised learning. Clustering algorithms group similar data points together based on their intrinsic characteristics, enabling researchers to discover inherent structures and segments within the data. Dimensionality reduction techniques, on the other hand, compress the data while retaining important information, making it easier to visualize and process large datasets. Unsupervised learning is particularly valuable in scenarios where obtaining labeled data is difficult or expensive, such as in market segmentation, anomaly detection, and recommendation systems.

2.1.1 Big Data Challenges

Big Data has emerged as a transformative force reshaping industries and driving innovation across various sectors. Its capacity to unlock valuable insights from vast and diverse datasets has the potential to revolutionize decision-making processes. Organizations can harness these insights to enhance customer experiences, optimize operations, and develop cutting-edge products and services. Researchers can gain a

deeper understanding of complex phenomena, such as climate change or disease outbreaks, by analyzing extensive datasets from multiple sources.

However, the journey towards realizing the benefits of Big Data is not without its challenges. One of the most significant hurdles lies in the sheer volume of data generated daily. This deluge of information necessitates advanced storage and processing solutions that can handle the immense scale of data, often pushing traditional infrastructure to its limits. The velocity at which data is generated further compounds this challenge, requiring real-time or near-real-time processing capabilities to extract timely insights.

Moreover, the variety of data types and sources, ranging from structured data in databases to unstructured data from social media and sensor networks, presents a formidable challenge in terms of data integration and harmonization. Ensuring that data from disparate sources can be effectively combined and analyzed is a complex task that requires careful consideration of data formats, semantics, and context.

Privacy and security concerns also loom large in the realm of Big Data. As the accumulation of sensitive information increases, the risk of data breaches and privacy violations becomes more pronounced. Striking a balance between utilizing data for insights while safeguarding individual privacy rights and complying with regulations like GDPR [27] or HIPAA [28] poses a significant challenge.

Furthermore, the quality of the data itself is a critical factor. Ensuring that the data used for analysis is accurate, reliable, and representative is essential to drawing meaningful conclusions. Poor data quality can lead to biased results or incorrect interpretations, undermining the credibility of insights.

In overcoming these challenges, technological advancements play a pivotal role. Machine Learning and Artificial Intelligence offer the potential to automate and streamline data processing, enhance data quality, and mitigate privacy risks through anonymization and differential privacy techniques.

In essence, Big Data's journey from raw information to actionable insights is marked by hurdles that demand innovative solutions. As technology continues to evolve, these challenges can be addressed, paving the way for a future where Big Data's transformative potential can be fully realized across industries and domains.

2.1.2 Applications of Machine Learning

Machine learning finds applications across a wide range of industries and sectors. This section provides an overview of the diverse domains where ML has made significant contributions.

In healthcare [29, 30, 31], ML is instrumental in disease diagnosis, where it can analyze medical data such as images, scans, and patient records to assist in early and accurate disease detection. ML also plays a significant role in drug discovery by identifying potential drug candidates and predicting their efficacy, thereby speeding up the drug development process. Furthermore, ML enables personalized medicine by leveraging genetic, clinical, and lifestyle data to tailor treatment plans for individual patients, optimizing their outcomes.

In the finance sector [32, 33, 34], ML techniques are widely used for fraud detection. These algorithms continuously analyze transaction data to detect unusual patterns or anomalies, helping financial institutions prevent fraudulent activities. ML also aids in risk assessment, as models assess borrowers' creditworthiness and portfolio risk, allowing for more informed lending decisions and investment strategies. Additionally, algorithmic trading relies on ML to make split-second decisions based on market data, optimizing trading strategies and enhancing returns for investors.

In the realm of marketing [35, 36, 37], ML is employed for customer segmentation, categorizing customers into groups based on their behavior. This enables businesses to tailor marketing campaigns and product recommendations for each segment, enhancing customer engagement and sales. Recommendation systems, powered by ML, provide personalized suggestions to users, as seen in platforms like Amazon and Netflix. Furthermore, ML is used for sentiment analysis, where algorithms process social media and customer feedback to gauge public sentiment, helping companies understand their brand perception and make data-driven marketing decisions.

In transportation [38, 39, 40], ML is at the forefront of innovation. Autonomous vehicles rely on ML to perceive their environment, make decisions, and navigate safely. ML models also predict traffic patterns and congestion, aiding commuters in planning their routes and helping cities manage traffic flow more efficiently. Additionally, logistics and supply chain operations benefit from ML, as it predicts demand, optimizes routes, and manages inventory more efficiently, reducing costs and delivery times.

These applications merely scratch the surface of ML's potential in various industries. Its capacity to derive insights from data, automate decision-making, and adapt to changing conditions makes it a transformative force, with its impact continuously expanding as technology advances.

2.2 Distributed Computing

Distributed computing refers to the use of multiple computers or nodes interconnected over a network to solve complex computational problems or process large-scale datasets. It enables parallel processing, efficient resource utilization, fault tolerance, and scalability, making it an indispensable approach for handling Big Data and computationally intensive tasks. This section provides a contextualization of distributed computing, discussing its key principles, architectures, challenges, and applications.

2.2.1 Principles of Distributed Computing

Distributed Computing represents a fundamental shift in the way we approach computational tasks, guided by a comprehensive set of principles that define its architecture and operation. Scalability, as a pivotal principle, addresses the need for systems to adapt seamlessly to varying workloads and data volumes. By distributing tasks across a network of interconnected nodes, distributed computing can efficiently handle larger and more demanding computational requirements, ensuring that performance remains robust as demands grow.

Fault tolerance is a critical facet of Distributed Computing, as it acknowledges the inherent challenges of operating in a networked environment where node failures can occur. Robust fault tolerance mechanisms, such as redundancy, replication, and failover strategies, safeguard against disruptions and data loss, ensuring that the system remains operational even in the presence of component failures.

Interoperability highlights is the importance of enabling different hardware and software components to communicate effectively within a distributed system. This principle ensures that diverse technologies can work cohesively, facilitating seamless data exchange, resource sharing, and collaboration among various nodes.

Resource sharing and allocation play a key role in optimizing the utilization of computational resources within a distributed environment. By dynamically allocating resources based on demand and priorities,

distributed computing systems can efficiently allocate processing power, memory, and storage, enhancing overall system performance and responsiveness.

Synchronization emerges as a crucial principle for managing concurrent processes and maintaining data consistency across distributed nodes. Effective synchronization mechanisms prevent conflicts and data corruption, ensuring that operations proceed in an orderly and coordinated manner.

Security and privacy are paramount in distributed computing, particularly given the increased exposure of data and communication across networked nodes. Strong encryption, authentication protocols, and access controls are essential to safeguard sensitive information and maintain the integrity of data flows within the distributed system.

Collectively, these principles shape the foundation of Distributed Computing, enabling the development of complex and robust applications that harness the collective power of interconnected machines. As technology continues to evolve, Distributed Computing remains a versatile and indispensable approach to addressing the challenges of a data-driven and interconnected world.

2.2.2 Architectures of Distributed Computing

Distributed computing architectures refer to the ways in which computational tasks are organized and executed across multiple interconnected computers or nodes. These architectures are designed to achieve better performance, scalability, fault tolerance, and resource utilization. Here are some of the common distributed computing architectures, along with their principal features, advantages, and disadvantages:

Client-Server Architecture

In a Client-Server architecture, as it shows in fig. 2, computing tasks are divided between clients and a central server. Clients, which can be devices or software applications, send requests to the server for services or resources. The Server processes these requests, performs necessary computations, and sends back the results or data. This architecture is well-suited for scenarios where there are many clients that need to access a centralized data source or application. It simplifies management and control since all data and services are managed on the server. However, it has limitations such as a single point of failure – if the server experiences issues, all clients may lose access. Additionally, scalability can be restricted by the Server's processing capacity.

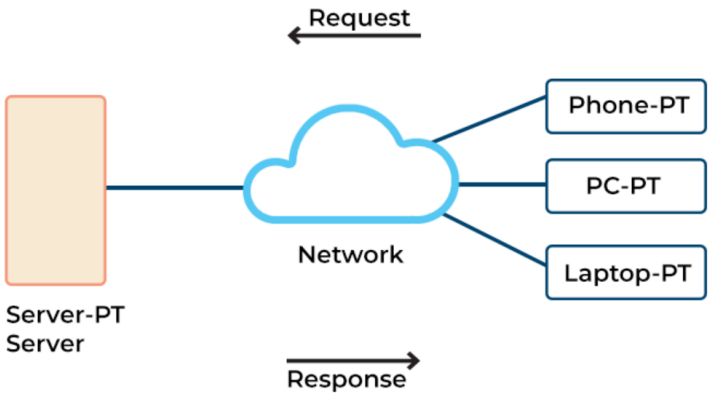


Figure 2: Client-Server architecture.

Peer-to-Peer (P2P) Architecture

In a peer-to-peer architecture, as depicted in 3, individual nodes (devices or computers) are interconnected and can act as both clients and servers. Each node can share its own resources, such as processing power or data, directly with other nodes in the network. This decentralized approach allows for greater fault tolerance since there is no single point of failure. It also offers scalability, as adding more nodes can increase the network's capacity. However, managing security and maintaining efficient communication between nodes can be challenging. Furthermore, as the network size grows, the performance of individual nodes may degrade due to increased demands on resources.

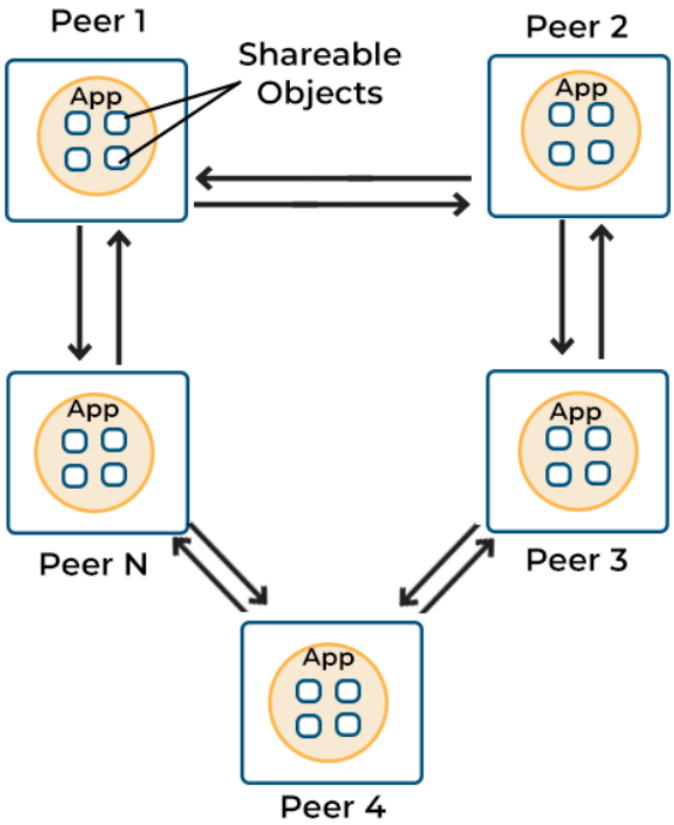


Figure 3: Peer-to-Peer architecture.

Cluster Computing Architecture

Cluster Computing involves creating a group of interconnected computers, as can be seen in fig. 4, known as a cluster, that work together to solve computational tasks. There are two main types of clusters: High-Performance Computing (HPC) clusters [41] and Load Balancing clusters [42]. HPC clusters are designed for compute-intensive tasks and scientific simulations, offering high computational power. Load Balancing clusters distribute tasks among nodes to optimize resource utilization and ensure efficient workload distribution. Clusters provide scalability by allowing more nodes to be added, which can enhance performance. However, setting up and configuring a cluster can be complex, and specialized hardware is often required, making it relatively costly.

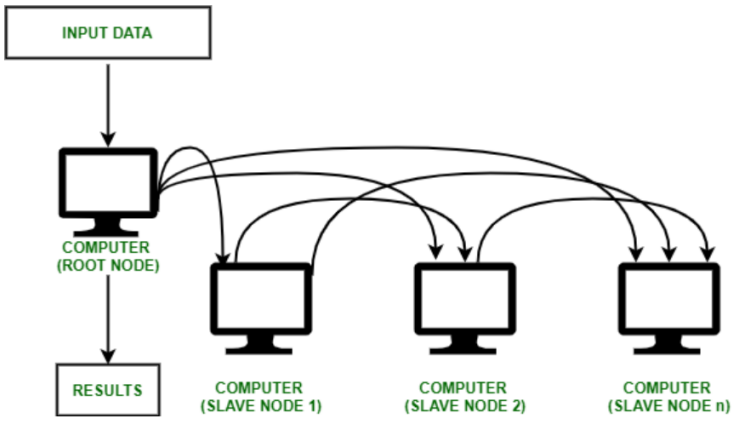


Figure 4: Cluster Computing architecture.

Grid Computing Architecture

Grid Computing, as depicted in fig 5, connects geographically distributed resources from different organizations to create a virtual computing environment. Resources can include computing power, storage, and data. Grids enable the aggregation of resources for tackling large-scale and complex problems. This architecture enhances fault tolerance and redundancy, as tasks can be rerouted to other resources in case of failures. However, managing a diverse and distributed environment with varying hardware and software configurations can be complex. Ensuring security and authentication across different organizational boundaries is also a significant challenge.

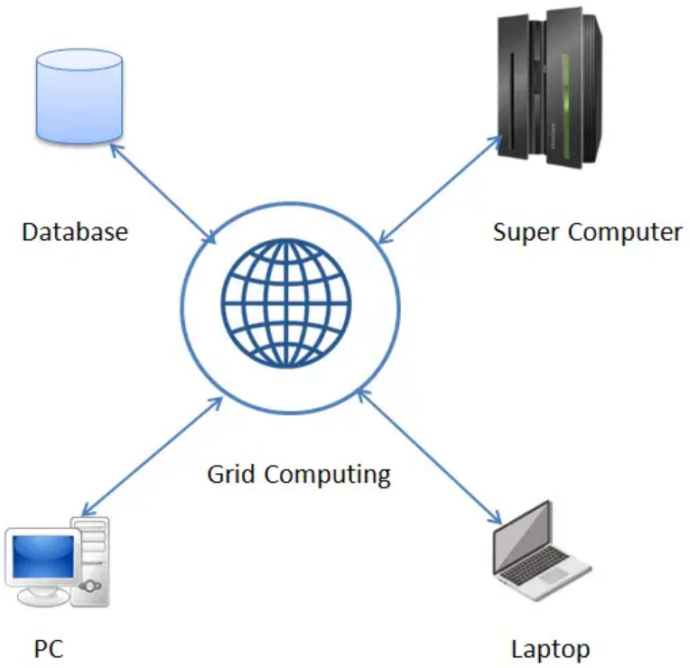


Figure 5: Grid Computing architecture.

Cloud Computing Architecture

Cloud computing, according to the depiction in fig. 6 provides on-demand access to a pool of configurable computing resources (such as virtual machines, storage, and services) over the internet. It offers scalability by allowing users to provision and scale resources as needed. This architecture is flexible, as users can choose from different service models (IaaS, PaaS, SaaS) [43] based on their requirements. Cloud computing eliminates the need for upfront hardware investments and provides cost-efficiency. However, it comes with dependencies on the service provider's infrastructure and performance. Concerns about data privacy, security, and compliance are important considerations when using cloud services.

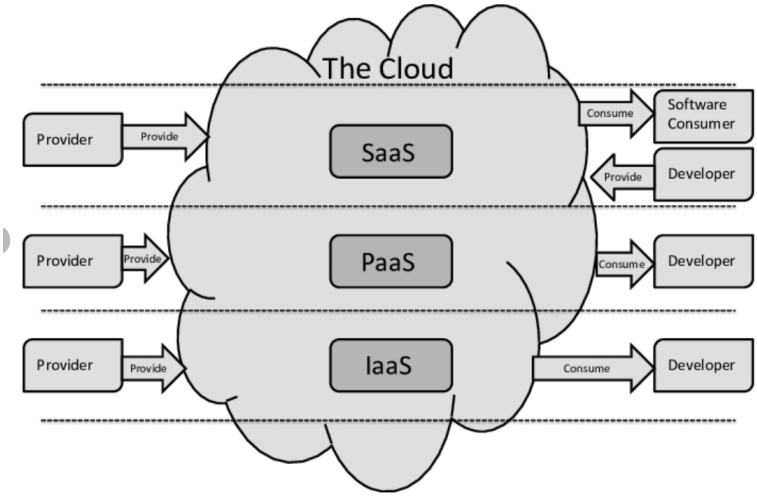


Figure 6: Cloud Computing architecture.

Fog/Edge Computing Architecture

Fog/Edge computing extends cloud computing capabilities closer to the data source or end-users, as it shows in fig. 7, reducing latency and enabling real-time processing. This architecture is particularly beneficial for applications like IoT [44], where timely data processing is essential. It offers improved response times and data privacy by processing data locally, at the edge of the network. However, edge resources are often limited in terms of processing power and storage compared to cloud environments. Managing distributed resources and ensuring consistent performance across different edge devices can be complex and challenging.

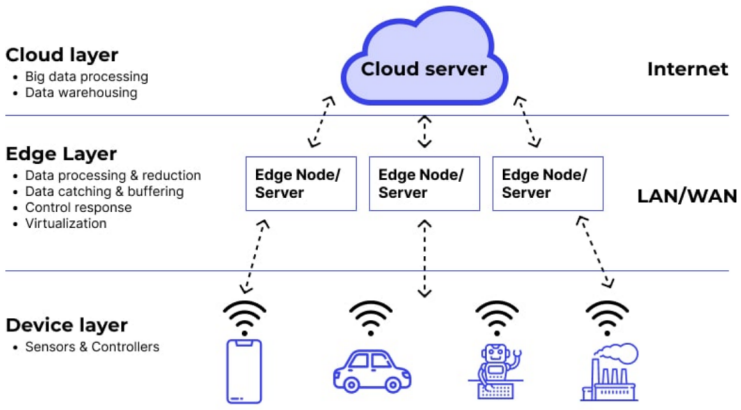


Figure 7: Edge Computing architecture.

Each of these Distributed Computing architectures has its own set of advantages and disadvantages, and the choice of architecture depends on the specific requirements, goals, and constraints of the application or system being developed. In many cases, hybrid or combined architectures are used to leverage the strengths of multiple approaches and mitigate their weaknesses.

2.2.3 Challenges in Distributed Computing

Distributed Computing, a paradigm that underpins modern systems and applications, presents a complex landscape of challenges and intricacies. One of the foremost challenges lies in achieving distributed consensus, where nodes in a network must collectively agree on a single value despite potential failures and communication delays. Distributed consensus algorithms like the classic Paxos [45] or the more recent Raft [46] are designed to address this challenge by ensuring that nodes reach agreement even in the presence of failures. However, these algorithms often require careful tuning and understanding of network conditions to achieve optimal performance.

Another significant hurdle is the management of distributed locks, essential for coordinating access to shared resources in a distributed system. Ensuring that multiple nodes do not simultaneously modify a shared resource, thus avoiding conflicts and maintaining data integrity, is a complex task. Distributed locking mechanisms, such as the Two-Phase Locking protocol [47] or distributed lock managers [48], offer solutions, but they introduce their own performance overhead and potential deadlocks if not managed skillfully.

Moreover, achieving fault tolerance and resilience is an ongoing struggle in Distributed Computing. Systems must be designed to gracefully handle node failures, network partitions, and other unexpected events. Techniques like replication, where copies of data are stored across multiple nodes, can help maintain availability even in the face of failures. However, the trade-offs between consistency, availability, and partition tolerance (CAP theorem) [49] often require careful consideration and trade-offs in system design.

Scalability, both in terms of performance and administration, is another challenge. As the number of nodes in a distributed system grows, the complexity of managing and coordinating these nodes increases exponentially. Ensuring that the system can efficiently scale up or down while maintaining consistent performance and low-latency communication demands intricate architectural choices.

The issue of data consistency is central as well. In distributed systems, maintaining a consistent view of data across nodes can be difficult due to the inherent delays in communication. Concepts like eventual consistency, where different nodes might observe different states of the system for a period of time, become relevant. Striking a balance between strong consistency and the performance benefits of relaxed consistency models requires careful tuning and understanding of application requirements.

Security and privacy concerns add another layer of complexity. Distributed systems are often more susceptible to attacks, such as Distributed Denial of Service (DDoS) [50], data breaches, and unauthorized access. Ensuring data encryption, secure communication channels, and robust access control mechanisms becomes paramount in such environments.

Lastly, debugging and monitoring distributed systems pose unique challenges. Identifying the root cause of an issue that spans multiple nodes can be extremely challenging. Distributed tracing and monitoring tools are essential, but they must be well-integrated and capable of handling the intricate interactions within the system.

In conclusion, the realm of Distributed Computing is rife with challenges that range from achieving consensus and managing locks to fault tolerance, scalability, data consistency, security, and monitoring. Addressing these challenges necessitates a deep understanding of both theoretical concepts and practical implementation intricacies, as well as a willingness to continually adapt and innovate as technology and systems evolve.

2.2.4 Applications of Distributed Computing

Distributed computing has widespread applications across various domains. Some notable examples include:

- Big Data Analytics: Distributed Computing frameworks like Apache Hadoop and Apache Spark enable processing and analysis of massive datasets, facilitating tasks such as data mining, machine learning, and real-time analytics.
- Cloud Computing: Cloud platforms leverage distributed computing to provide scalable and on-demand resources to users, enabling services like virtual machines, storage, and serverless computing.
- Internet of Things (IoT): The IoT relies on distributed computing to process and analyze data generated by interconnected devices, enabling applications like smart cities, industrial automation, and healthcare monitoring systems.
- Scientific Simulations: Distributed Computing allows scientists to perform large-scale simulations and computational experiments, aiding research in fields such as physics, climate modeling, and molecular dynamics.

2.3 Distributed Machine Learning

Distributed Machine Learning (DML) is an amalgamation of machine learning and distributed computing, aimed at addressing the challenges posed by large-scale datasets and computationally intensive tasks. By harnessing the power of distributed systems, DML enables the training and inference of machine learning models across multiple interconnected nodes. This section provides an overview of Distributed Machine Learning, discussing its principles, architectures, algorithms, and applications.

2.3.1 Principles of Distributed Machine Learning

Distributed Machine Learning (DML) embodies a set of principles that leverage the power of Distributed Computing to tackle the formidable challenges posed by training large-scale and complex machine learning models. At its core, DML capitalizes on the idea of distributing the computational workload across multiple machines or nodes, thereby enabling the processing of massive datasets and intricate models that might otherwise be infeasible to handle on a single system. One fundamental principle is data parallelism, where the dataset is divided into subsets that are distributed among nodes, each performing local computations and contributing to the overall model update. This approach not only expedites the training process but also enables the utilization of resources more efficiently.

Another key tenet is model parallelism, which comes into play when a single machine lacks the memory capacity to accommodate the entire model. In this scenario, the model is divided into segments that are processed by different nodes, effectively allowing for the training of large and complex models by combining the individual updates from each segment. However, coordinating these updates and maintaining the model's integrity across different nodes necessitate careful synchronization and communication mechanisms, highlighting the importance of efficient communication frameworks such as parameter servers.

Consensus and coordination mechanisms are vital to DML as well, ensuring that the model parameters converge to a consistent state across all nodes. Distributed consensus algorithms like all-reduce [51] and parameter server architectures facilitate synchronization among nodes, enabling them to agree upon the optimal model parameters despite the communication delays and potential failures inherent to distributed systems. Additionally, fault tolerance and resiliency remain significant principles in DML. Systems must be designed to handle node failures, network partitions, and communication bottlenecks gracefully, guaranteeing the continuity of the training process without significant disruptions.

Resource allocation and management form another cornerstone of DML principles. Dynamic allocation of computational resources ensures that each node contributes optimally to the training process, thereby enhancing the overall efficiency of the distributed system. Techniques like elastic scaling enable nodes to be added or removed dynamically based on the computational demands, optimizing resource utilization without compromising on performance.

Ensuring privacy and security is also of paramount importance in DML, especially when dealing with sensitive data. Principles of differential privacy, secure multi-party computation, and federated learning are employed to safeguard individual data points while still enabling collaborative model training across distributed nodes.

In conclusion, the principles of Distributed Machine Learning revolve around harnessing the capabilities of distributed computing to surmount the challenges posed by large-scale and intricate machine learning tasks. Data parallelism, model parallelism, consensus mechanisms, fault tolerance, resource management, privacy, and security considerations collectively define the foundations of DML, enabling the development of robust and scalable machine learning systems that can process vast datasets and tackle complex models effectively.

2.3.2 Architectures of Distributed Machine Learning

In the realm of DML, diverse architectural strategies play a pivotal role in addressing the complexities of training intricate models on extensive datasets across distributed environments.

The concept of Data Parallelism forms a foundational architecture wherein the training dataset is partitioned into segments, with each segment distributed across various nodes or workers. Data Parallelism proves especially efficacious when model parameters are shared among nodes and each node can perform computations autonomously. This approach expedites training by parallelizing the processing of distinct data segments, leading to quicker model convergence. Nonetheless, with an increasing number of nodes or training iterations, communication overhead might rise, necessitating well-thought-out communication strategies and synchronization mechanisms.

In the context of models that are too large to fit into the memory of a single machine, the Model Parallelism architecture becomes indispensable. This architecture involves segmenting the model into parts or layers, with each segment assigned to a separate distributed node. Nodes independently compute the outputs of their allocated model segments using their local data. These outputs are then transmitted across nodes for gradient calculations, which are subsequently employed to update the model parameters. The Model Parallelism strategy effectively addresses memory constraints, thereby enabling the training of complex models that would otherwise be unmanageable. Seamless communication and synchronization mechanisms, however, emerge as critical components to ensure a coherent model representation across distributed segments.

The Parameter Server architecture addresses the challenge of communication efficiency inherent to distributed training setups. Nodes in this architecture are divided into two distinct roles: workers and parameter servers. Workers process local data and compute gradients based on their computations. These gradients are then communicated to parameter servers, which undertake the centralization of the management and storage of global model parameters. Parameter servers subsequently disseminate updated model parameters back to workers for successive training iterations. This architecture optimizes communication efficiency by facilitating a centralized mechanism for parameter updates, thereby minimizing communication overhead. Careful considerations regarding load balancing and the avoidance of bottlenecks at parameter servers are pivotal to achieve optimal system performance.

For scenarios necessitating privacy preservation and data decentralization, the Federated Learning architecture is particularly pertinent. This approach extends the training process to edge devices such as smartphones or IoT devices. In lieu of transmitting raw data to a central server, these devices locally train models using their respective data and send only the model updates to a central server. Aggregation of these updates at the central server results in the creation of a global model that encapsulates insights from all participating devices. Federated Learning ensures data privacy by retaining raw data on individual devices, thereby sharing only aggregated updates. This architecture proves invaluable when centralizing data is impractical or introduces privacy risks.

Furthermore, Hybrid Architectures merge elements of these architectures, often targeting optimization, challenge-specific solutions, and adaptability to particular demands. By leveraging the strengths of various paradigms, hybrid approaches cater to distinct facets of the training process, thereby offering an adaptive toolkit that aligns with diverse training objectives.

To encapsulate, these architectures in Distributed Machine Learning offer a spectrum of strategies for designing efficient, scalable, and privacy-conscious systems capable of training complex models across distributed environments. The choice of architecture hinges upon factors such as dataset size, model intricacy, communication patterns, resource constraints, and privacy considerations. Each architecture contributes uniquely to overcoming specific challenges, ultimately advancing the landscape of distributed model training.

2.3.3 Distributed Machine Learning Algorithms

The algorithms of Distributed Machine Learning (DML) constitute a cornerstone of designing efficient and scalable systems for training complex models across distributed environments. These algorithms encompass a range of strategies that tackle challenges such as communication bottlenecks, synchronization, fault tolerance, and convergence in the distributed training process. One prominent class of algorithms is Distributed Gradient Descent, which includes variants like Downpour SGD [52] and Bulk Synchronous Parallel (BSP) [53].

These algorithms synchronize model updates at specific intervals, allowing nodes to perform independent computations on their local data subsets before aggregating gradients and updating the global model. Another significant algorithmic approach is Asynchronous Stochastic Gradient Descent (ASGD) [54], which allows nodes to update the model asynchronously, mitigating synchronization overhead. However, ASGD introduces challenges like potential staleness of model updates and order inconsistencies.

Moreover, Distributed Consensus Algorithms [55] such as the classic Paxos [45] and its derivatives, like Raft [56], are pivotal for ensuring that nodes converge to a common model despite network delays

and potential failures. These algorithms establish agreement protocols for achieving distributed consensus on model updates.

Federated Averaging [57], a central technique in Federated Learning, aggregates model updates from different devices with varying data distributions while considering the participation of each device. Additionally, Quantization and Compression Algorithms [58] are deployed to reduce the communication burden by compressing model updates before transmission, ensuring more efficient utilization of network resources. Algorithms like Elastic Averaging SGD [59] enable dynamic node participation, allowing nodes to join or leave the training process without interrupting global model convergence.

Finally, Byzantine Fault-Tolerant Algorithms [60] are crucial for countering malicious nodes that might intentionally provide erroneous updates. These algorithms ensure the integrity of model aggregation in the presence of Byzantine nodes. Collectively, these algorithms underscore the complexity and ingenuity required to navigate the intricate landscape of Distributed Machine Learning, facilitating the design of systems that can effectively address challenges arising from distributed computations, communication constraints, privacy concerns, and system failures.

2.3.4 Applications of Distributed Machine Learning

The application of Distributed Machine Learning (DML) is a transformative force across a spectrum of industries and domains, redefining the way complex models are trained and insights are extracted from vast and intricate datasets.

In the realm of healthcare [61], DML enables the analysis of medical records and images across distributed hospitals, facilitating the development of predictive models for disease diagnosis and treatment recommendations while ensuring patient data privacy through techniques like Federated Learning.

In the financial sector [62], DML empowers institutions to detect fraudulent activities by training models on distributed transaction data, enhancing security while preserving sensitive customer information.

In the field of autonomous vehicles [63], DML plays a pivotal role in training sophisticated perception models by aggregating data from various sensors across a fleet of vehicles, thereby enhancing real-time decision-making capabilities while ensuring collective knowledge sharing.

In the agricultural sector [64], DML leverages data from distributed sensors to create models that optimize irrigation, pest control, and crop yield prediction, enabling sustainable farming practices.

Additionally, DML is revolutionizing scientific research, such as particle physics, by enabling collaborative model training across multiple institutions for processing data from large-scale experiments like the Large Hadron Collider. The application of DML extends to natural language processing, where models trained on distributed datasets facilitate language translation, sentiment analysis, and text generation.

Overall, the application of Distributed Machine Learning transcends industries, offering solutions that enhance efficiency, accuracy, privacy, and collaboration while navigating the intricacies of large-scale distributed systems and empowering advancements across a multitude of sectors.

2.4 Distributed Machine Learning Platforms

The landscape of Distributed Machine Learning Platforms has ushered in a transformative era in the field of artificial intelligence, redefining the boundaries of what's achievable in model complexity and data scale.

These platforms harness the power of Distributed Computing to overcome the limitations of traditional Machine Learning setups, enabling the efficient processing of vast datasets and the training of intricate models across multiple nodes or machines. Within this rich ecosystem, a multitude of platforms have flourished, each distinguished by its unique capabilities and strengths.

TensorFlow [65], a creation of Google, boasts a comprehensive ecosystem facilitating the construction and training of Machine Learning models at scale, while PyTorch has gained traction for its dynamic computational graph and seamless integration with Python, simplifying complex tasks.

The Hadoop [66] ecosystem, coupled with Apache Spark, stands as a formidable framework for distributed data processing and Machine Learning tasks. Alongside these giants, Microsoft's Azure Machine Learning [67] and Amazon Web Services' SageMaker [68] offer cloud-based solutions that streamline the deployment and management of machine learning workflows.

H2O [69], an open-source framework, excels in providing a scalable environment for building and training models on distributed clusters. H2O was the platform selected to compare with the presented solution and it will be discussed in more detail in the next section.

Furthermore, Databricks cater to specialized needs, showcasing the diversity within the distributed machine learning realm. This expansive array of platforms signifies not only the versatility of distributed machine learning but also its potential to democratize advanced AI techniques, empowering researchers, developers, and organizations to harness the distributed paradigm for groundbreaking advancements in Artificial Intelligence applications.

2.5 H2O

H2O, a standout gem within the constellation of distributed machine learning platforms, has emerged as a potent force in the advancement of artificial intelligence. This open-source framework offers a dynamic and scalable environment that empowers practitioners to construct and train complex machine learning models on distributed clusters. H2O is celebrated for its versatile capabilities and user-friendly interface, making it an attractive choice for data scientists and engineers alike. Supporting multiple programming languages such as Python, R, and Java, H2O facilitates seamless integration into existing workflows. One of its defining features is the integration with popular big data processing frameworks like Hadoop and Apache Spark, allowing it to efficiently handle large datasets that might otherwise pose challenges for conventional machine learning platforms.

At the heart of H2O's power lies the concept of the H2O cluster, a dynamic ensemble of interconnected computing nodes that work in harmony to execute machine learning tasks. The H2O cluster leverages distributed computing paradigms, effectively distributing data and computations across multiple machines for accelerated performance and increased scalability. Each node in the cluster contributes processing power, memory, and storage, creating a collective computational powerhouse that can handle substantial workloads with efficiency. This cluster architecture is especially advantageous when dealing with resource-intensive tasks like model training on massive datasets or hyperparameter tuning across a wide parameter space.

The H2O cluster management system ensures seamless coordination between nodes, allowing them to communicate, synchronize, and optimize tasks collaboratively. This orchestration minimizes bottlenecks and maximizes resource utilization, resulting in faster model training times and improved overall perfor-

mance. Furthermore, H2O's cluster setup offers fault tolerance, ensuring that if a node fails, the tasks can be seamlessly rerouted to other nodes without disrupting the entire process.

In essence, the H2O cluster embodies the core ethos of distributed machine learning – harnessing the combined strength of multiple machines to tackle challenges that would be insurmountable for a single machine. As the landscape of artificial intelligence continues to evolve, H2O's emphasis on scalability, efficiency, and versatility positions it as a pivotal tool in enabling researchers, developers, and organizations to explore the frontiers of AI through the lens of distributed computing.

2.6 Funtional emphasis

This project boasts several remarkable features that set it apart from H2O. One of the standout capabilities of this project is its ability to construct diverse ensembles in real-time while employing the same base models, facilitated by an advanced optimization module. This dynamic feature allows users to adapt their ensemble models on-the-fly, tailoring them to specific tasks or adapting to changing data patterns. In contrast, H2O is characterized by its rigidity, lacking the flexibility to build such real-time ensembles with the same models, making the project particularly valuable for tasks requiring agile and adaptive model performance.

A defining characteristic of this project is its capacity to create heterogeneous ensembles, a capability that H2O does not possess. The project in question enable users to combine a variety of diverse models, each with distinct strengths and weaknesses, into a single ensemble. This amalgamation of heterogeneous models results in a powerful ensemble that can outperform individual models by capitalizing on their complementary abilities. This stands in contrast to H2O, which typically involves ensembles based on homogeneous models, limiting the range of techniques that can be integrated into the ensemble. The heterogeneous ensemble capability of this work provides practitioners with a novel tool for tackling complex and multifaceted problems effectively.

Moreover, this project empowers users with a heightened level of control over the system compared to H2O. The project offers a comprehensive suite of customization options, allowing users to fine-tune various aspects of the ensemble-building process. This includes parameter adjustments for individual models, ensemble composition strategies. In contrast, H2O might provide limited options for ensemble customization, potentially restricting the extent to which users can adapt the system to their specific needs. The increased control offered by this work not only caters to the expertise of seasoned machine learning practitioners but also encourages experimentation and innovation by granting them the ability to craft ensembles that align precisely with their objectives.

In summary, this project presents a compelling suite of features that distinguishes it from H2O. The project's real-time ensemble building capabilities using the same models through an optimization module, heterogeneous ensemble creation, and the granular control granted to users collectively contribute to this work's versatility and effectiveness in various machine learning tasks. While H2O serves as a valuable tool in its own right, this work's unique strengths make it a particularly appealing choice for those seeking advanced ensemble modeling techniques with enhanced adaptability and control.

Chapter 3

Architecture

This chapter is divided in two parts. In the first section, the proposed architecture of the developed system will be described. In the second section, it will be explained how the implemented system works internally, the way the used technologies were implemented and how they communicate with each other.

3.1 Proposed architecture

The entire system is included in a docker cluster, and in this system we can have as many node containers as it can be managed, fig. 8 shows that there may be n worker nodes . To the specific purpose of this project were only implemented three worker nodes. Three were used because it represents the minimum number of containers needed to test the replication and parallelism functionalities. In a real case scenario, the more nodes, the greater the data partitioning and the more use is made of parallelism. The coordinator container, which contains the user interface and the REST API. The coordinator is also responsible for sending the tasks created by users to the worker nodes to execute. Then the containers that pertain to each worker node and finally, a container for mlflow that has the sole function of storing the performance metrics of each base model of each project.

Users make use of the interface to create or execute tasks and the interface communicates with the API to record those tasks on a MongoDB database. In the case of executing tasks such as training and model prediction, the moment the user makes the request for task execution, the API communicates with the coordinator node via ZeroMQ providing all the necessary data for workers to be able to perform the tasks without any problems. In turn, the coordinator communicates with the worker nodes through a publish/subscribe protocol, forwarding all the information for training or prediction models.

As the worker nodes finish the tasks to which they have been assigned, they send the results to the coordinator node, which is listening via a push/pull protocol implemented in ZeroMQ. Once the coordinator receives the results generated by the workers, it operates on that data to obtain the information it wants and stores that information in HDFS (e.g. predictions of each base model).

3.2 Functional Description

In this section a more technical description will be given of how the architecture works and how the respective components communicate with each other. A more in-depth explanation of the system flow

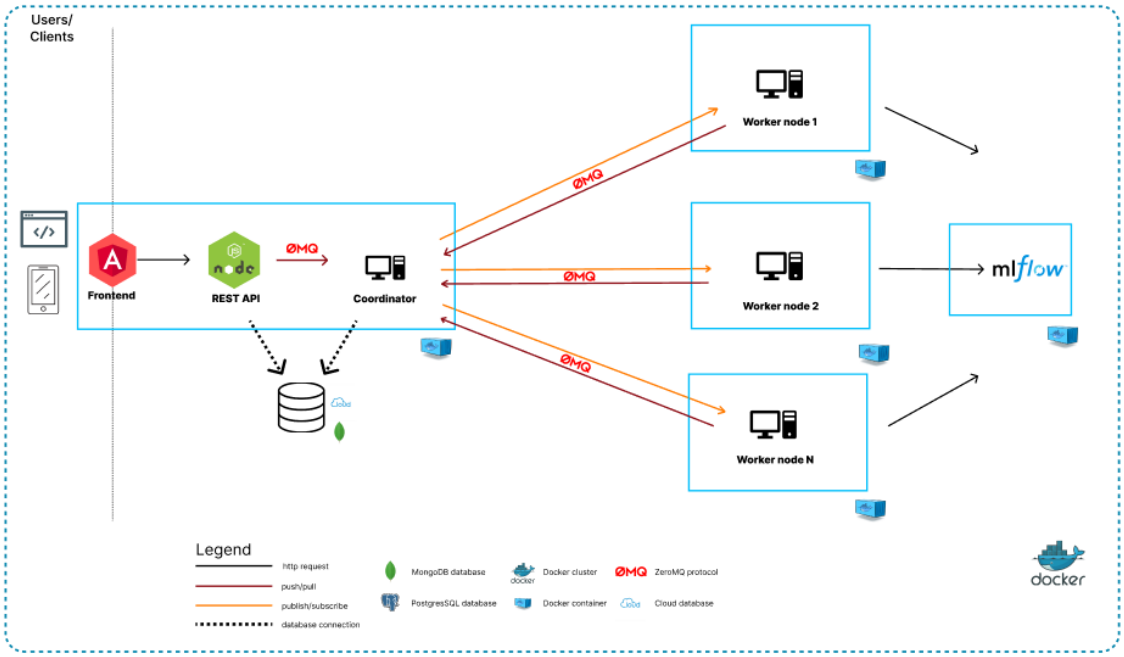


Figure 8: System's proposed architecture.

will also be given, presenting which data is generated and its importance, also referring to some specific functionalities of the system.

3.2.1 Data locality and HDFS

Data locality is a fundamental concept in distributed computing, and its relationship with HDFS is particularly significant. HDFS is designed to store and manage large-scale datasets across a cluster of machines, and data locality plays a crucial role in optimizing the performance of data processing tasks. In the context of this project, data locality manifests itself when the coordinator possesses information about which nodes hold each block of data and can strategically assign a specific node to operate on a block it already contains. This approach eliminates the need for data transfer, as the computation is moved to where the data resides.

In a typical HDFS setup, data is divided into fixed-size blocks shown in fig. 17 that are distributed across the cluster. Each block is replicated on multiple nodes to ensure fault tolerance. The fig. 18 shows the configuration for block size and replication factor. When a data processing task, such as a Spark job, is initiated, the coordinator determines the optimal assignment of tasks to nodes. This decision is guided by the principle of data locality, aiming to minimize network communication and maximize the utilization of local resources.

In CEDEs, the coordinator, equipped with knowledge of data block placement, can effectively schedule computations on the nodes that already possess the required data blocks. By doing so, data transfer overhead is significantly reduced or eliminated altogether. This approach capitalizes on the principle of data locality, leveraging the proximity of computation to data to enhance efficiency and performance.

By utilizing data locality, this project demonstrates a smart utilization of the HDFS infrastructure. Instead of moving the data across the network to a particular node for processing, the system takes advantage of the existing distribution of data blocks and distribute the computation to the nodes that hold

the relevant data. This strategy not only saves network bandwidth and reduces latency but also enables efficient parallel processing of data, allowing for scalable and high-performance analysis.

Overall, by leveraging data locality in this system, there is a possibility to harness the full potential of HDFS, optimizing resource usage and reducing unnecessary data transfers. This approach aligns with the core principles of distributed computing and contributes to the efficient processing of large-scale datasets.

3.2.2 Communication Protocols

There are two essential protocols implemented in ZeroMQ that allow the system to function, one being the protocol that allows the coordinator to send information to the worker nodes, and the other being the protocol that allows the workers to send the results obtained to the coordinator.

In the communication from the coordinator to the workers, the implemented protocol uses a publish/subscribe pattern. In this pattern the workers, when connected, are listening, and any exchange of information by the coordinator is sent in broadcast to all connected nodes. In this case, the coordinator sends the necessary information so that the workers can carry out the operations without any problems. The information that the coordinator sends to the workers depends of three scenarios (training, predicting file, predicting line).

The information that is sent for training is:

- The action to the workers carry out (Train or Predict).
- The name of the project chosen by the user.
- The header of the dataset.
- The variables that the user wants to drop, for example if there are variables that are useless for the problem or variables that as little to no information that can help to build models with better performance.
- The dependent variable of the problem. This is the variable that is going to be predicted.
- The machine learning problem type. (Binomial classification, Multinomial classification or Linear regression).
- The size of the test set. A 0.25 value of this variable represents that 25% of data is reserved exclusively for testing and the remaining 75% for training.
- The percentage of the machine learning algorithms that will be used for training. For example, a value of (50%, 20%, 30%) represents that each model has 50% of probability to be trained as a Random Forest, 20% as a Decision Tree and 30% as a Neural Network.
- The list of the blocks to be trained.
- Algorithm hyper-parameters (optional).

The information that is sent for predicting a file is:

- The action to the workers carry out (Train or Predict).

- The name of the project to be used for prediction task.
- Hashmap with the list of blocks of data each worker will predict.
- Hashmap with the sub-models each workers must use for prediction task.
- Name of the folder to save the predictions.

The information that is sent for predicting a line is:

- The action to the workers carry out (Train or Predict).
- The name of the project to be used for prediction task.
- Line of data to be predicted.
- Hashmap with the sub-models each workers must use for prediction task.

In the communication from workers to the coordinator, a push/pull protocol was implemented. In this protocol, the coordinator is receiving all the messages that are sent to a certain port and then receives the results of all the workers as they arise. As the workers finish training each sub-model, they send information about it to the coordinator. The information that the workers send to the coordinator depends of two scenarios (training, predicting).

The information that is sent for training is:

- Performance metrics of the respective sub-model.
- Duration of training of the respective sub-model.
- Processing time of the data to train.
- The name of the project to associate the metrics.
- The id of the block corresponding to the trained sub-model.
- The IP of the worker in which the sub-model was trained.
- The training date.
- The algorithm used for training that specific sub-model.

The information that is sent for predicting is:

- Information on whether it is the forecast for one line or several.
- The predictions (whether it's a single line or several).

3.2.3 Data preprocessing

In this subsection, is discussed the process of preprocessing data and developing encoders to transform non-numerical values into numerical representations. To achieve this, we designed and implemented an application in Spark that operates on the dataset after it is uploaded to HDFS.

The preprocessing phase is crucial in data analysis as it involves transforming raw data into a format suitable for machine learning algorithms. In our case, the dataset consists of non-numerical values, such as categorical variables, which need to be encoded numerically to enable their utilization in various statistical models. By leveraging the power of Spark, we can efficiently process large-scale datasets in a distributed manner, ensuring scalability and performance.

The preprocessing application in Spark applies a series of transformations and encodings to the dataset. It starts by loading the dataset from HDFS into Spark's distributed memory, enabling parallel processing across the cluster. Then, it performs a range of preprocessing steps, such as handling missing values, scaling features, and normalizing data if required.

A key aspect of the preprocessing application is the creation of encoders. These encoders are responsible for converting non-numerical values into numerical representations that machine learning algorithms can understand. For example, categorical variables can be encoded using techniques like one-hot encoding, label encoding or target encoding. The choice of encoding technique depends on the nature of the data and the specific requirements of the analysis.

Once the preprocessing steps and encoding processes are completed, the application generates a JSON file that contains all the necessary information for subsequent stages of the data analysis pipeline. This JSON file serves as a crucial artifact, as it encapsulates the transformations applied to the dataset and the encoding schemes adopted for different variables.

The generated JSON file can then be utilized during training and prediction processes. It provides the necessary information to apply the same preprocessing and encoding steps consistently to new data instances, ensuring compatibility and consistency across different stages of the analysis. By utilizing this preprocessed data, we can focus on building robust and accurate machine learning models without worrying about the intricacies of handling non-numerical values.

3.2.4 Model training

As mentioned above, all datasets are stored in HDFS and are divided into blocks of a predefined size. Therefore, given this partition of data, the system creates an ensemble, where the sub-models trained on the basis of each partition of this data integrate the ensemble. When the user decides to create an ensemble, the system requests some information from the user, including the project name, the name of the dataset to be used for training, the header with all the variables in the dataset (dependent and independent), whether or not to remove any variables that are deemed irrelevant for model training, the dependent variable, the type of problem (binary classification, multinomial, or regression), the test size, the percentage of creation of each model (e.g. 50%, 30%, 20% means that our ensemble will be composed by 50% of Random Forests, 30% of Decision Trees and 20% of Neural Networks), and the respective model configurations (optional). In this case, since each model in the ensemble is subject to a probability of being trained using a different algorithm, we can create heterogeneous ensembles.

After the user provides the requested information, the coordinator, using the dataset name, retrieves the list of blocks for the workers to train the respective sub-models. Using ZeroMQ, the coordinator sends

an JSON object to all workers containing the previously defined configurations. The models are allocated in the HDFS in a folder with the project name that the user defined.

In order for the worker nodes to know which nodes to train, there is a Hashtable where the key is the IP of the workers and the value is the list of blocks that each worker should operate on. If the number of blocks is less than the number of active nodes, only the first node will train the list of blocks. However, this is a rare case since the system aims to work with large volumes of data.

Once the worker nodes receive the information from the coordinator, the training process begins. Each worker node accesses the HDFS folder where the blocks from its list are stored and performs the training one by one. First, the data to be trained is prepared. The data is read using pandas (a file reading module), by creating a dataframe with the block data. The user-provided header is added, redundant variables for training are removed. The data is then divided into two parts: features and labels, where features are the independent variables and labels are the dependent variable of the dataset. Furthermore, the data is split into training and test data, where the training data is used to train the sub-model and the test data is used to make predictions for calculating performance metrics.

Finally, the sub-model is trained, and associated performance metrics are calculated and stored in a dictionary. At the end of training each sub-model, the sub-model is stored in the HDFS and submitted at its replication funcionality which allows different workers to use the models for prediction purposes. Furthermore, each worker node sends the performance metrics and training duration associated with the blocks it trained to the coordinator through ZeroMQ. The coordinator stores this information in a JSON file for future analysis.

Fig. 9 presents a sequence diagram that represents the entire process described above.

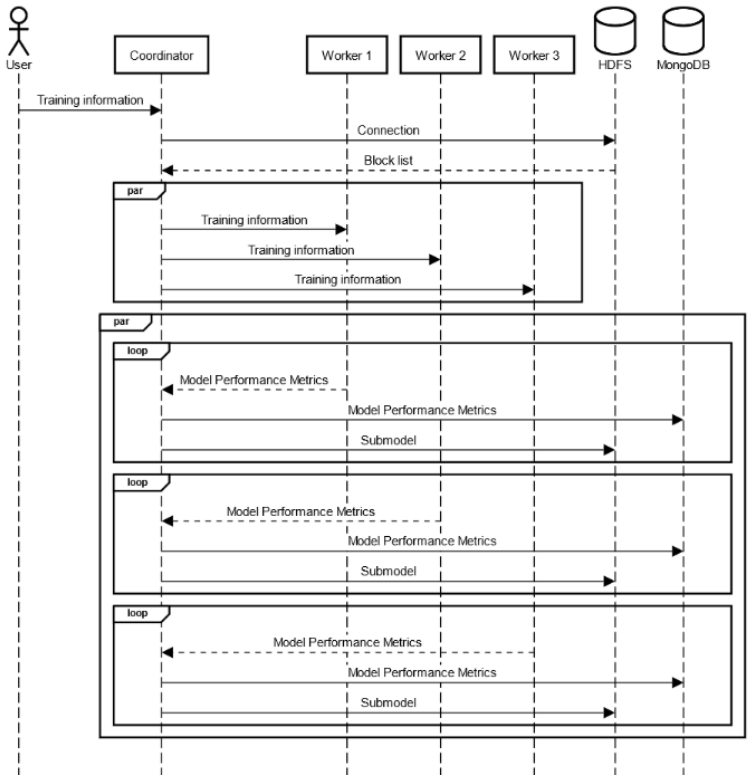


Figure 9: Model training sequence diagram.

3.2.5 Model prediction

In the case of model prediction, the process ends up being very similar to the process of training models. As mentioned above, the models trained for a particular project that is related to a particular machine learning problem are also allocated in HDFS so that, in the prediction process, the file system's replication functionalities can also be used to make predictions in parallel in an optimised way, so that there are several candidate nodes for using a particular sub-model to make predictions.

There are two ways of making predictions in the system. The first and simplest way is to predict a line of data. In this case, the user only needs to provide the line of data they want to forecast and the project they want to use to make the predictions. The system automatically obtains the sub-models associated with that project and makes the predictions for each sub-model for the respective data line in parallel. As the predictions are generated, the worker node responsible for the sub-model that made the prediction sends it to the coordinator. When all the sub-models have generated their prediction for the data line and the coordinator receives the respective prediction, the coordinator averages or fashions the predictions in order to obtain a generalised prediction value for that specific line.

The second and slightly more complex way is to predict a file previously uploaded to HDFS. In this case, the file is divided into blocks and in order for all sub-models to be able to predict each block of the file, there is a need for all sub-models related to the problem to be predicted to have a replication factor equal to the number of active worker nodes in the system. This way, all the workers can use all the sub-models to make predictions. Thus, each worker uses all the sub-models to make predictions for a given block. When all the sub-models have made their predictions, an average is made for each line of the predicted block and the dataframe with the average of the predictions is sent to the coordinator. When the workers have finished predicting all the blocks in parallel, the coordinator puts all the blocks together in a file and saves the results of the predictions in the HDFS.

Fig. 10 represents the sequence diagram of the functionality of predicting a file previously uploaded to HDFS.

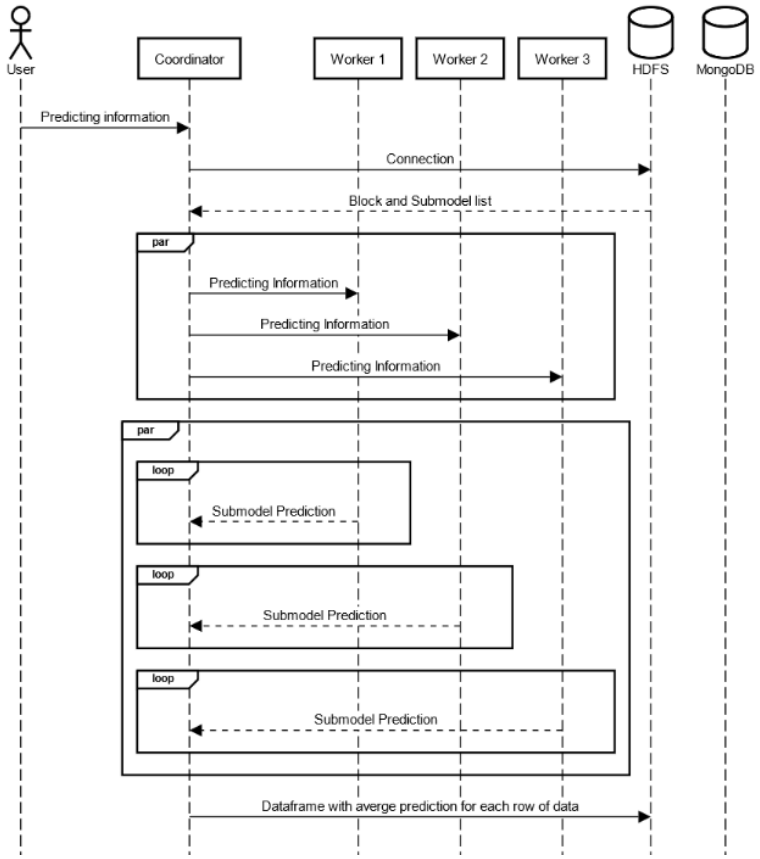


Figure 10: Model predicting sequence diagram.

Chapter 4

Validation and Results

4.1 Problem Statement

This project shed light on intriguing conflicts within distributed systems. These conflicts often arise from either the configuration options of the utilized tools or the decisions made during software implementation. This dissertation focuses on one such conflict, and the following section outlines the methodology employed to address the unanswered questions.

A crucial issue revolves around the block size, a configuration parameter of the distributed file system used. In HDFS, the block size directly impacts file storage. A smaller block size results in smaller blocks that can be balanced and replicated more quickly across the cluster. However, it also introduces a significant overhead in terms of file system management, as the metadata stored for each block remains independent of the block size. Consequently, a smaller block size necessitates maintaining a larger number of blocks, leading to increased overhead. Notably, HDFS's main limitation lies in memory: it performs better with a few large files rather than numerous small ones.

However, the implications for this system extend beyond this point. Since there exists a one-to-one relationship between blocks and base models, a smaller block size implies more blocks and, consequently, more base models. In theory, these base models will be smaller, simpler, and weaker in terms of predictive capabilities since they were trained with less data. As the performance of an Ensemble is measured by averaging the performance of its base models, weaker models result in weaker Ensembles. While a smaller block size enhances parallelism to a certain extent by allowing more base models to be trained in parallel at a lower cost, it also increases the overhead related to coordination and task distribution.

Conversely, employing a larger block size theoretically improves the performance of both base models and the Ensemble, minimizes coordination overhead, but reduces the degree of parallelism. Moreover, if the block size is sufficiently large that only a few blocks exist, the advantages of Ensemble methods are forfeited, potentially resulting in an Ensemble with just a single model.

This work aims to address this problem by presenting and discussing it first, and then determining the optimal point that maximizes parallelism and model quality. It is important to note that this analysis should be conducted on a case-by-case basis, considering the number of available nodes in the cluster. In the following section, we provide a generic evaluation.

Additionally, another objective is to assess different heuristics for creating Ensembles, which is carried out in the subsequent section. Specifically, we evaluate and compare three potential heuristics. Since these heuristics are user-selectable when configuring the ML project in the system, it is crucial to compare their

performance.

4.2 Materials and Methods

The previous section described the main problem addressed by this work. This section details the methodology followed in its analyzis (fig. 11). Four different publicly available datasets were used in the experiments conducted. A characterization of these datasets is provided in Table 1.

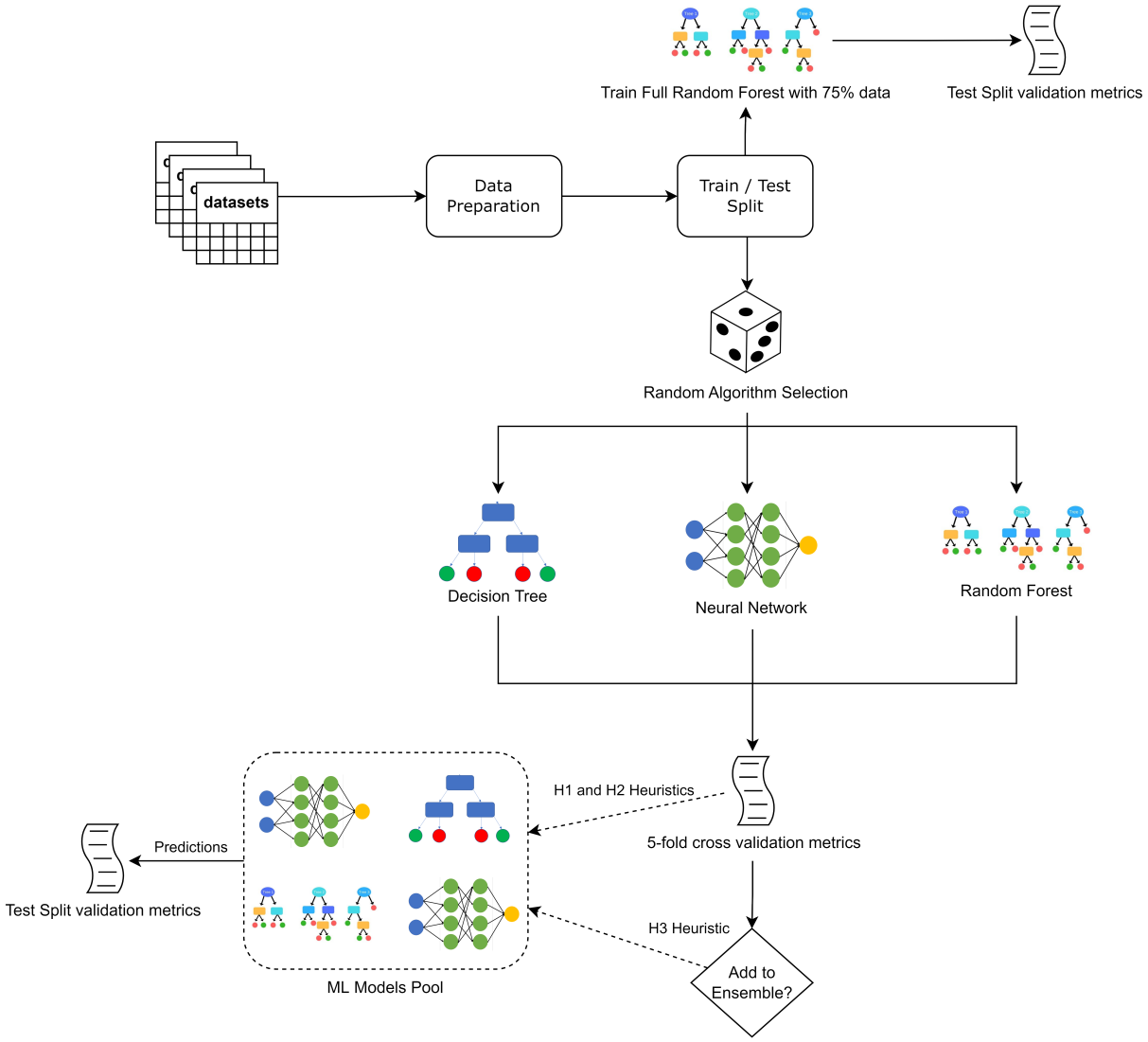


Figure 11: Methodology followed in each of the experiments.

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 initial datasets were acquired and subjected to preprocessing. This involved various tasks such as encoding specific variables and eliminating outliers. Outliers were identified and removed from the original dataset using the 1.5 times the interquartile range (IQR) rule. The IQR is calculated using Equation (4.1), where q_{75} and q_{25} denote the third and first quartiles, respectively.

$$iqr = q_{75} - q_{25} \tag{4.1}$$

Then, the lower and upper limits for removing outliers are given by Equations (4.2) and (4.3), respectively.

$$lower = q_{25} - 1.5 \times iqr \tag{4.2}$$

$$upper = q_{75} + 1.5 \times iqr \tag{4.3}$$

Next, each dataset is filtered by the dependent variable, as given by Equation (4.4) in which x denotes each instance of dataset X.

$$filtered = \{x \in X \mid x < upper \ \wedge \ x > lower\} \tag{4.4}$$

The preprocessed datasets were uploaded to the HDFS, followed by a series of ML experiments conducted as outlined below. Initially, the datasets were shuffled and divided into training and testing subsets, using a 75/25 ratio. The training sets were utilized to train various Ensembles, while the held-out test sets were solely employed for evaluating the Ensembles' generalization performance.

Throughout the experiments, the HDFS was sequentially configured with different block sizes, denoted as $B = [4, 8, 16, 32, 64]$ (in MB). Consequently, each dataset was split into varying numbers of blocks based on their respective sizes.

For each block size and dataset, three distinct heuristics (referred to as H_1 to H_3) for constructing Ensembles were tested, which will be elaborated upon shortly. Hence, a total of 60 Ensembles were trained, comprising 5 block sizes multiplied by 4 datasets multiplied by 3 heuristics. Notably, these Ensembles possessed different numbers of base models, contingent upon the chosen block size.

The algorithm for each base model was selected randomly, while adhering to the standard configurations provided in Table 2. In the case of Neural Networks, the activation functions were also chosen randomly. Consequently, the created Ensembles exhibited heterogeneity, and if the process were repeated, the results might exhibit slight variations due to the random algorithm selection. However, given the dataset's size, these variations were expected to be insignificant.

This project utilizes the scikit-learn library, specifically employing three algorithms in this study: Random Forest (sklearn.Ensemble.RandomForestClassifier), Decision Tree (sklearn.tree.DecisionTreeClassifier),

and Neural Network (sklearn.neural_network.MLPClassifier). The three tested heuristics operate 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 [70]) is:

$$\text{RMSE}(y, \hat{y}) = \sqrt{\frac{\sum_{i=0}^{N-1}(y_i - \hat{y}_i)^2}{N}} \tag{4.5}$$

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:

$$W_m = \frac{\sum_{i=1}^{n} \epsilon_i, i \neq m}{\sum_{i=1}^{n} \epsilon_i} \times \frac{1}{n-1} \tag{4.6}$$

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).

Throughout this process, similar to the development of this project, we acknowledge the possibility of uneven data distribution. This implies the presence of trends, variations, and the potential for items within different blocks (or even within the same block) to be drawn from diverse probability distributions. Consequently, the performance of base models within an Ensemble may exhibit substantial fluctuations based on their input blocks. These variations are explored by the Optimization module, which employs various criteria to construct the most optimal Ensemble, such as utilizing a subset of the best-performing models.

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

$$\forall i \neq j \; p(x^{(i)}, x^{(j)}) = p(x^{(i)})p(x^{(j)}) \tag{4.7}$$

In the context of this project, it is important to note that the used datasets do not exhibit any inherent relationships between instances, including temporal or order relationships. While these types of datasets can be utilized in this system, it is worth mentioning that the algorithms implemented thus far may not be the most suitable for handling such tasks. To summarize, this work was specifically designed and validated for scenarios involving independent non-identically distributed data [71].

During the training process of the base models, their quality is assessed using the RMSE in two ways. Firstly, the RMSE is calculated through 5-fold cross-validation, providing an evaluation within the training process itself. Additionally, since there is a dedicated test set for each dataset, the RMSE is also computed on the test set to evaluate the generalization ability of the models.

This evaluation process is significant for determining the extent to which the cross-validation RMSE serves as a reliable indicator of future model quality. In H_1, the cross-validation RMSE is used to determine the model's weight, while in H_3, it is utilized to select the top 50% of models. A strong correlation between these two RMSE values would suggest that the assumption of using cross-validation RMSE as a reliable indicator of model quality is valid. However, in scenarios where the test data significantly differs from the training data, such as cases involving concept drift, this assumption may not hold true. A low correlation would indicate such a discrepancy.

To explore this further, the following section compares the cross-validation RMSE with the RMSE on the test data. Additionally, the RMSE values for each block size and heuristic are analyzed to determine the optimal configuration for this system.

4.3 Results

In the previous section, we examined the correlation between the RMSE of Ensembles calculated through cross-validation and the RMSE calculated on the test data. In this section, it is important to note that when discussing or analyzing individual RMSE values (as depicted in the following figures), we present the absolute values of RMSE. However, when comparing the performance of different models across various datasets, we express it as a percentage of the range of values of the dependent variable. This approach eliminates the influence of the RMSE's dependence on the scale of the dependent variable, allowing for a fair comparison between different machine learning problems.

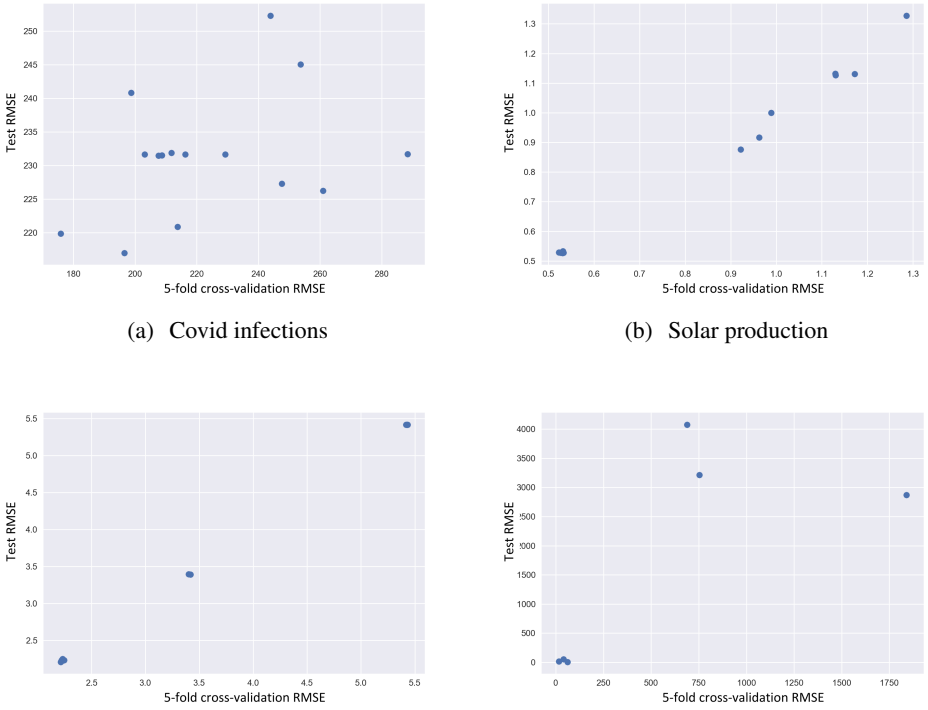
Dataset	**4 MB**	**8 MB**	**16 MB**	**32 MB**	**64 MB**
Covid infections	0.79	0.66	0.63	0.35	0.31
Solar production	0.93	0.94	0.99	0.99	0.98
Temperature	0.99	0.99	0.99	0.99	0.99
Car prices	0.29	0.25	0.43	0.71	0.99

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

The analysis of Table 3 reveals two interesting trends. In the Covid infections dataset, there is a tendency for the correlation to weaken as the block size increases. On the other hand, in the remaining three datasets, the correlation generally strengthens as the block size increases. This latter trend aligns

with our expectations and intuition, indicating that smaller block sizes may result in base models with seemingly good performance but limited ability to generalize to new data.

To visualize this information for the 32 MB block size, fig. 12 has been provided. Each dot on the graph represents a single base model. The x-axis represents the RMSE measured through 5-fold cross-validation, while the y-axis represents the RMSE measured on the test set.



(a) Covid infections

(b) Solar production

(c) Temperature

(d) Car prices

Figure 12: 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).

In addition to the Ensembles trained following the methodology described in the previous section, a baseline comparison was conducted by training a Random Forest for each dataset using the entire dataset. The RMSE of the Random Forest models was evaluated through 5-fold cross-validation.

Comparing the RMSE directly across different models and problems is not appropriate due to the dependence of RMSE on the scale of the dependent variable. To ensure meaningful and generalizable conclusions, an analysis was performed using the RMSE divided by the range of the outlier limits of the dependent variable. These outlier limits were calculated using the 1.5 IQR rule. This approach provides the error as a percentage of the range of values.

Upon analyzing Tables 4-6, a general observation can be made that as the block size increases, the error decreases. However, it is important to note the trade-off involved: larger block sizes result in reduced parallelism as there are fewer available nodes. Therefore, finding the optimal block size is crucial to strike a balance between maximizing parallelism and minimizing error metrics.

The Covid infections dataset presents the greatest challenge for both the implemented system and a standard Random Forest due to its specific characteristics as a time-series dataset. Time-series forecasting,

which involves seasonality, trends, and cyclical variations, often requires more specialized algorithms such as ARIMA [72] or LSTM [73].

The distinct nature of the Covid infections dataset becomes evident when analyzing the correlations between cross-validation and test errors. In all other datasets, the correlation strengthens with increasing block size, indicating better generalization as more data is included in each block. However, the Covid dataset exhibits the opposite trend.

In terms of model performance, the baseline Random Forest, that was trained using H2O platform, achieves an error of 6.68% (of the range of dependent variable values). This system Ensembles, on the other hand, yield errors ranging from 7.4% (H_1, 64 MB) to 8.02% (H_3, 4 MB). Although the baseline Random Forest outperforms our Ensembles in every scenario, the results are close.

The remaining datasets generally yield better results for both the baseline Random Forest and our Ensembles.

In the Solar production dataset, the baseline Random Forest achieves an error of 0.08%. Similarly to the Covid infections dataset, the baseline Random Forest outperforms the Ensembles created by the system in all tested scenarios. The best Ensemble for this dataset is obtained with H_3 and a block size of 32 MB, resulting in an error of 0.41%. The worst performance occurs with H_2 and a block size of 16 MB, yielding an error of 0.67%.

The situation differs in the other two datasets, where our approach surpasses the baseline Random Forest.

For the Temperature dataset, the baseline Random Forest has an error of 0.078%. In all 15 scenarios tested, this system outperforms the baseline Random Forest. The best performance is achieved with H_3 and a block size of 32 MB, resulting in an error of 0.022%. The worst performance occurs twice, with H_1 and H_2 using a block size of 4 MB, yielding an error of 0.0317%.

Regarding the Car prices dataset, the baseline Random Forest achieves an error of only 0.001%. The best Ensemble obtained by this system surpasses it with an error of 0.0002%. This is achieved with H_3 and a block size of 32 MB. The worst performance is observed with H_2 and a block size of 4 MB, resulting in an error of 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.

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

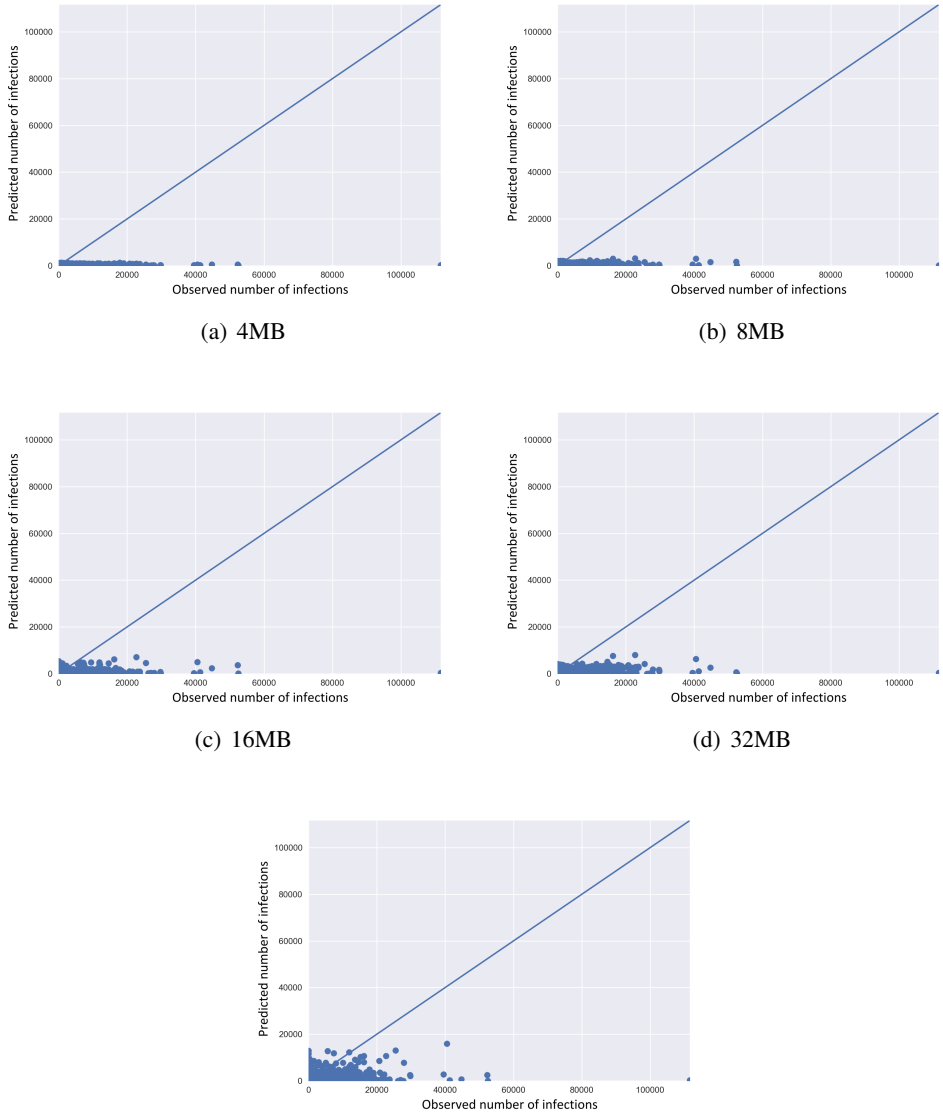
Table 6: RMSE in percentage of the scale of the dependent variable, for each block size and heuristic H_3.

The following content present a visual analysis of the performance of selected Ensembles. Since H_3 has shown to be the best performing heuristic (i.e., yielding the best results for the temperature and car prices datasets, even outperforming the baseline Random Forest), the analysis was focused on this heuristic to keep the paper concise.

Figs. 13 to 16 illustrate the results for the Covid infections, Solar production, Temperature, and Car prices datasets, respectively. Each figure displays the observed values on the x-axis and the corresponding predicted values by each Ensemble on the y-axis. The units of the axes correspond to those of the original target variable. The diagonal line represents perfect prediction, indicating that the closer the data points are to this line, indicating a higher similarity between the predicted and actual values.

As previously discussed, the Covid infections dataset exhibits the poorest performance, which is consistent with the baseline Random Forest. This suggests that the inherent complexity of the problem or the limited relevance of the variables in the dataset contribute to the challenging nature of this dataset.

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.



(a) 4MB

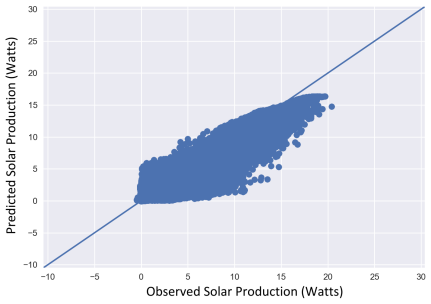
(b) 8MB

(c) 16MB

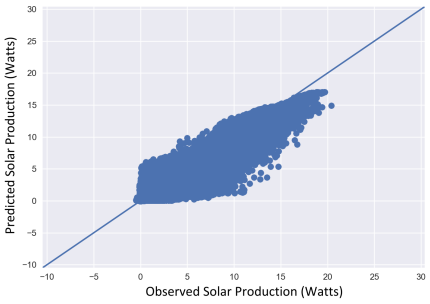
(d) 32MB

(e) 64MB

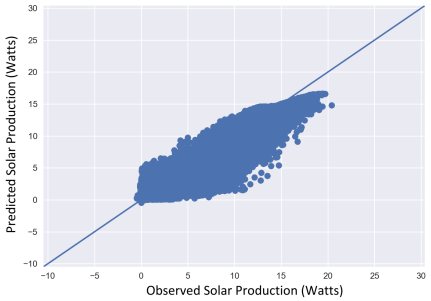
Figure 13: Predicted values vs. observed values (number of infections) for the Covid infections dataset, using different block sizes. The solid line represents the region of a perfect prediction. Base models for this time-series problem have generally poor performance since the algorithms used are not specific for time-series.



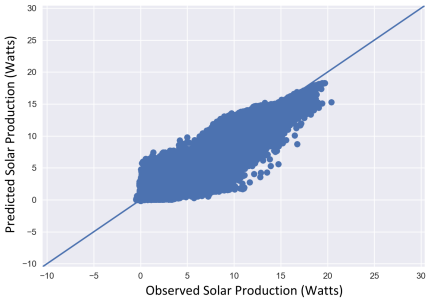
(a) 4MB



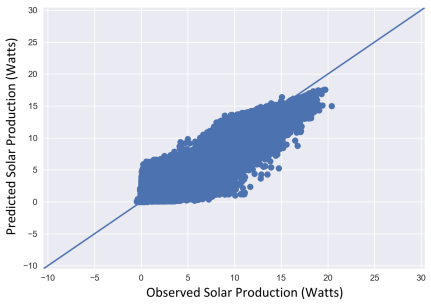
(b) 8MB



(c) 16MB

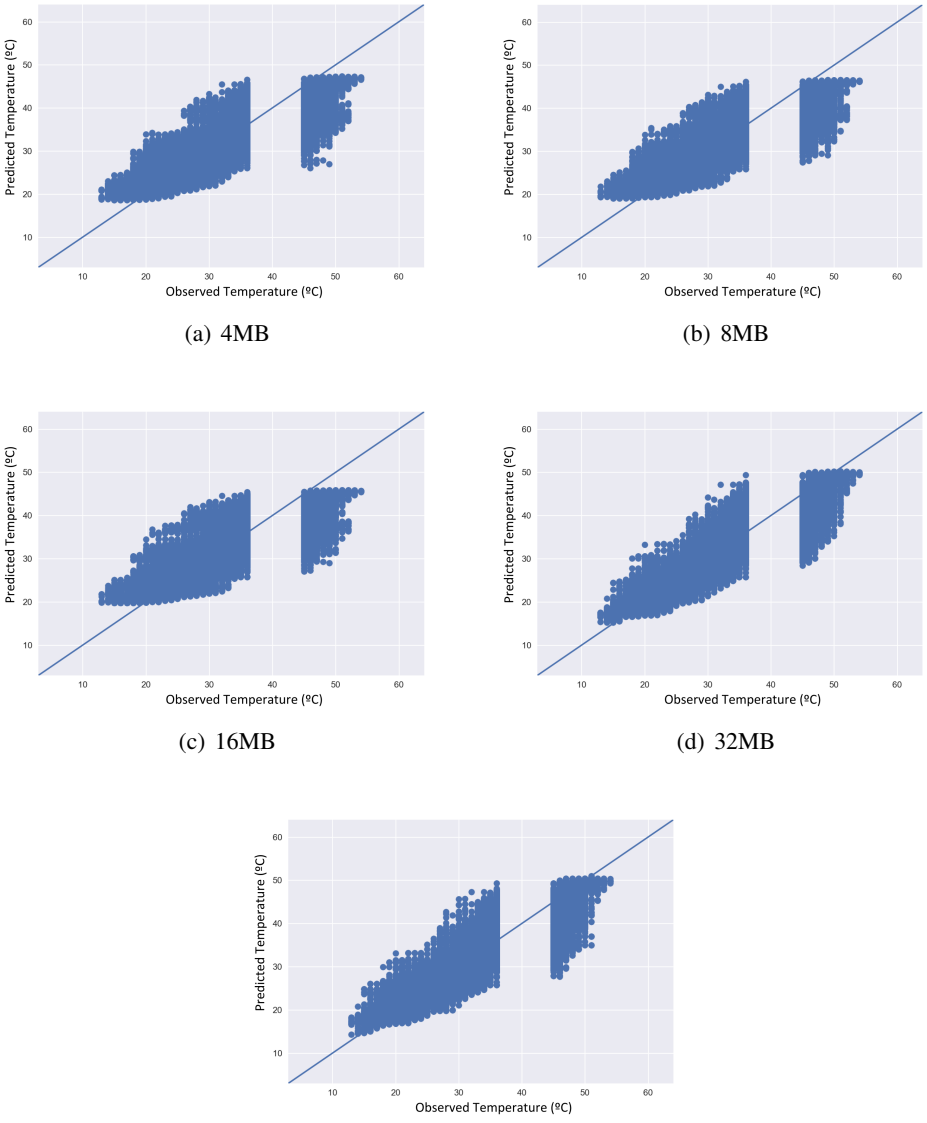


(d) 32MB



(e) 64MB

Figure 14: 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

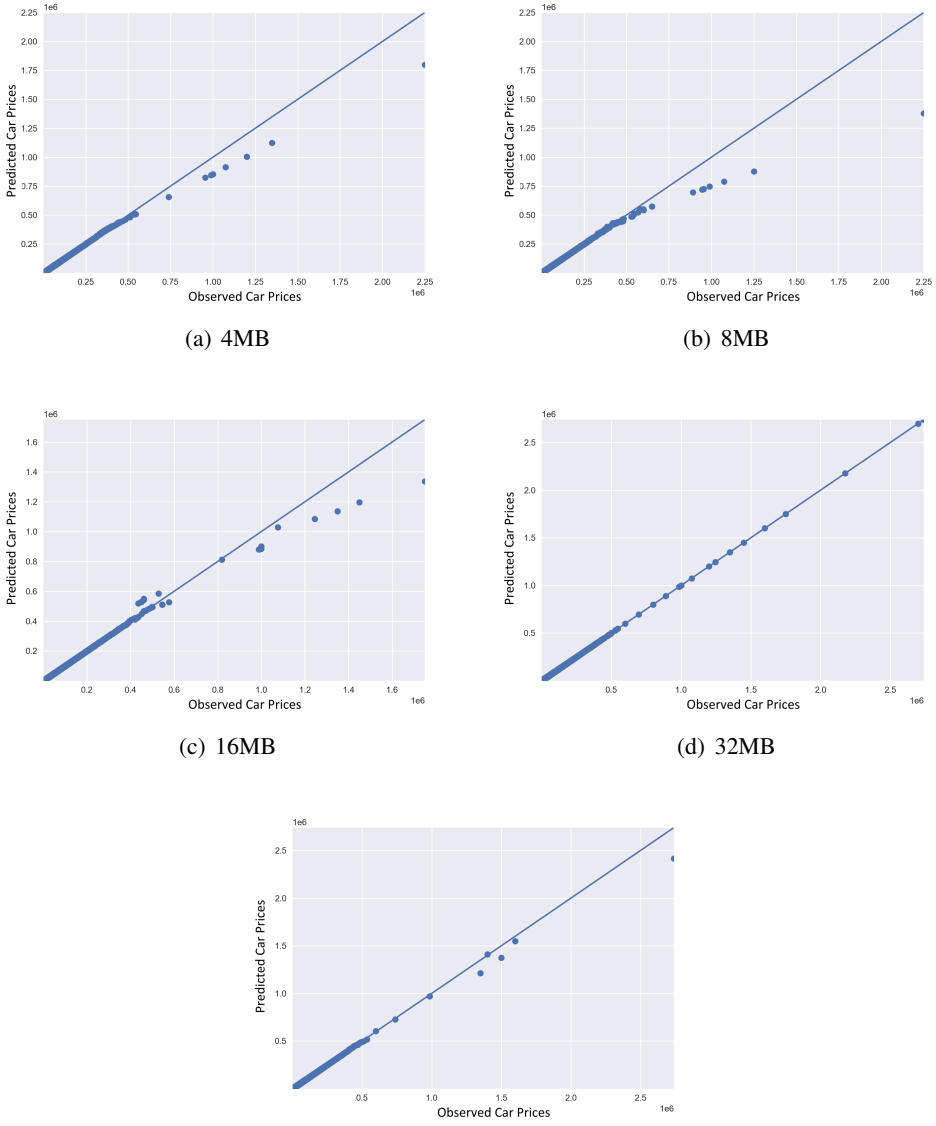
(b) 8MB

(c) 16MB

(d) 32MB

(e) 64MB

Figure 15: Predicted values vs. observed values for the Temperature dataset (ºC), 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 16: 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.

Chapter 5

Conclusions and Future Work

5.1 Conclusion

This dissertation introduces a distributed environment for Machine Learning applications, and highlights its innovative features. One key aspect of this system is the real-time assembly of Ensembles based on available base models and cluster state, making it dynamic and capable of updating Ensembles over time by adding or removing models. This functionality is facilitated by the optimization module, which is currently under development.

The primary objective of this paper was to investigate the impact of block size and different heuristics on the quality of the Ensemble. The study aimed to determine if a particular heuristic consistently outperforms others and identify an optimal block size.

Three heuristics were tested, as described in Section 4.2. Among these, H_3 demonstrated generally lower error rates. H_3 selects the best 50% of available base models, assigning equal weight to each model.

The results suggest that smaller Ensembles might outperform more complex ones, indicating that poorly performing base models could have a negative impact on Ensemble performance. This insight enables more efficient management of base models. Continuously underperforming models can be deleted and removed from the HDFS, freeing up resources.

H_3 did not perform better only in the case of the Covid infections dataset, which is a challenging dataset for the considered algorithms due to its time-series nature. Thus, this exception should not cast doubt on the suitability of H_3.

Another significant finding is that a block size of 32 MB consistently outperforms other options in all scenarios. This block size strikes a good balance between parallelism, coordination overhead, and Ensemble size. Smaller block sizes increase coordination overhead without significant performance gains, while larger block sizes offer reduced coordination overhead but yield inferior results. Consequently, the optimal block size is determined to be 32 MB.

It is important to note that the goal of this work was not to achieve the best possible results for each dataset but to identify the best heuristic and block size. Standard configurations were used for each algorithm, and better results could potentially be obtained by exploring alternative configurations through trial and error or using heuristics such as grid search.

Nonetheless, the results are considered satisfactory. This system outperforms a standard Random Forest in two out of four datasets (Temperature and Car prices), and the results are comparable for the other two datasets (Covid infections and Solar production).

Using Ensembles to implement heuristics offers advantages such as model adaptability and reduced risk of overfitting. However, it also sacrifices interpretability, as Ensembles become more complex and provide limited insights into feature-outcome relationships. Future work will integrate explainable AI [74] developments, such as model-agnostic approaches like LIME [75] or Shapley [76] values, into the system to address this limitation.

Another focus of future work will be detecting and mitigating bias in Ensemble models. Ensemble models have the potential to improve accuracy and robustness, but they can also amplify biases if base models exhibit similar biases. Ensuring diversity and complementarity among base models and implementing bias monitoring and prevention measures will be crucial.

To further expand the analysis, additional datasets and larger block sizes will be included. While the tested block sizes were relatively small compared to the standard HDFS block size of 128 MB, the results indicate that 32 MB is the optimal size. However, further exploration with diverse datasets will enhance confidence in this regard.

Based on the dissertation's findings, a model management module will be implemented to track the usefulness of each base model by monitoring its selection frequency in Ensembles. Base models that are not utilized by H_3 will be removed over time, freeing up cluster resources.

In conclusion, the work conducted allowed us to ascertain the ability of this system 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.

5.2 Scientific Results

5.2.1 Journal Publications

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

   Guimarães, M., Carneiro, D., Oliveira, F., Novais, P. (2023). Block Size, Parallelism and Predictive Performance: Finding the Sweet Spot in Distributed Learning. IJPEDS. https://doi.org/10.1080/17445760.2023.22

1. Predicting Model Training Time to Optimize Distributed Machine Learning Applications

   Guimarães M, Carneiro D, Palumbo G, Oliveira F, Oliveira Ó, Alves V, Novais P. Predicting Model Training Time to Optimize Distributed Machine Learning Applications. Electronics. 2023; 12(4):871. https://doi.org/10.3390/electronics12040871

5.2.2 Conference Publications

3. A Data-Locality-Aware Distributed Learning System

   Carneiro, D., Oliveira, F., Novais, P. (2022). A Data-Locality-Aware Distributed Learning System. In: Novais, P., Carneiro, J., Chamoso, P. (eds) Ambient Intelligence – Software and Applications – 12th International Symposium on Ambient Intelligence. ISAmI 2021. Lecture Notes in Networks and Systems, vol 483. Springer, Cham. https://doi.org/10.1007/978-3-031-06894-2_6

2. Dynamic Management of Distributed Machine Learning Projects

   Oliveira, F. et al. (2023). Dynamic Management of Distributed Machine Learning Projects. In: Braubach, L., Jander, K., Bădică, C. (eds) Intelligent Distributed Computing XV. IDC 2022. Studies in Computational Intelligence, vol 1089. Springer, Cham. https://doi.org/10.1007/978-3-031-29104-3_3

1. The impact of data selection strategies on distributed model performance

   Guimarães, M., Carneiro, D., Oliveira, F., Novais, P. (2023). The impact of data selection strategies on distributed model performance. ISAmI'23. https://link.springer.com/chapter/10.1007/978-3-031-43461-7_16

Acknowledgments

This work has been supported by FCT – Fundação para a Ciência e Tecnologia, through a research grant within project EXPL/CCI-COM/0706/2021.

Bibliography

[1] Zhi-Hua Zhou. *Machine learning*. Springer Nature, 2021.

[2] Aytuğ Onan. Bidirectional convolutional recurrent neural network architecture with group-wise enhancement mechanism for text sentiment classification. *Journal of King Saud University-Computer and Information Sciences*, 34(5):2098–2117, 2022.

[3] Aytuğ Onan. An ensemble scheme based on language function analysis and feature engineering for text genre classification. *Journal of Information Science*, 44(1):28–47, 2018.

[4] Aytuğ Onan. Sentiment analysis on product reviews based on weighted word embeddings and deep neural networks. *Concurrency and Computation: Practice and Experience*, 33(23):e5909, 2021.

[5] Lina Zhou, Shimei Pan, Jianwu Wang, and Athanasios V Vasilakos. Machine learning on big data: Opportunities and challenges. *Neurocomputing*, 237:350–361, 2017.

[6] Batta Mahesh. Machine learning algorithms-a review. *International Journal of Science and Research (IJSR).[Internet]*, 9:381–386, 2020.

[7] Rameshwar Dubey, Angappa Gunasekaran, Stephen J Childe, Constantin Blome, and Thanos Papadopoulos. Big data and predictive analytics and manufacturing performance: integrating institutional theory, resource-based view and big data culture. *British Journal of Management*, 30(2):341–361, 2019.

[8] Mehdi Mohammadi, Ala Al-Fuqaha, Sameh Sorour, and Mohsen Guizani. Deep learning for iot big data and streaming analytics: A survey. *IEEE Communications Surveys & Tutorials*, 20(4):2923–2960, 2018.

[9] Heitor Murilo Gomes, Jesse Read, Albert Bifet, Jean Paul Barddal, and João Gama. Machine learning for streaming data: state of the art, challenges, and opportunities. *ACM SIGKDD Explorations Newsletter*, 21(2):6–22, 2019.

[10] Chen Xu, Xiaoping Du, Xiangtao Fan, Gregory Giuliani, Zhongyang Hu, Wei Wang, Jie Liu, Teng Wang, Zhenzhen Yan, Junjie Zhu, et al. Cloud-based storage and computing for remote sensing big data: a technical review. *International Journal of Digital Earth*, 15(1):1417–1445, 2022.

[11] Zedong Tang, Fenlong Jiang, Maoguo Gong, Hao Li, Yue Wu, Fan Yu, Zidong Wang, and Min Wang. Skfac: Training neural networks with faster kronecker-factored approximate curvature. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13479–13487, 2021.

[12] Sylvie Retout, Stephen Duffull, and France Mentré. Development and implementation of the population fisher information matrix for the evaluation of population pharmacokinetic designs. *Computer methods and Programs in Biomedicine*, 65(2):141–151, 2001.

[13] Xibin Dong, Zhiwen Yu, Wenming Cao, Yifan Shi, and Qianli Ma. A survey on ensemble learning. *Frontiers of Computer Science*, 14:241–258, 2020.

[14] Hieu Pham and Sigurdur Olafsson. Bagged ensembles with tunable parameters. *Computational Intelligence*, 35(1):184–203, 2019.

[15] Rebecca Roelofs, Vaishaal Shankar, Benjamin Recht, Sara Fridovich-Keil, Moritz Hardt, John Miller, and Ludwig Schmidt. A meta-analysis of overfitting in machine learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[16] Smitha Rajagopal, Poornima Panduranga Kundapur, and Katiganere Siddaramappa Hareesha. A stacking ensemble for network intrusion detection using heterogeneous datasets. *Security and Communication Networks*, 2020:1–9, 2020.

[17] Sergio González, Salvador García, Javier Del Ser, Lior Rokach, and Francisco Herrera. A practical tutorial on bagging and boosting based ensembles for machine learning: Algorithms, software tools, performance study, practical perspectives and opportunities. *Information Fusion*, 64:205–237, 2020.

[18] Ji Liu, Jizhou Huang, Yang Zhou, Xuhong Li, Shilei Ji, Haoyi Xiong, and Dejing Dou. From distributed machine learning to federated learning: A survey. *Knowledge and Information Systems*, 64(4):885–917, 2022.

[19] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, pages 1–10. Ieee, 2010.

[20] Hagit Attiya. Concurrency and the principle of data locality. *IEEE Distributed Systems Online*, 8(9):3–3, 2007.

[21] Dastan Maulud and Adnan M Abdulazeez. A review on linear regression comprehensive in machine learning. *Journal of Applied Science and Technology Trends*, 1(4):140–147, 2020.

[22] Barry De Ville. Decision trees. *Wiley Interdisciplinary Reviews: Computational Statistics*, 5(6):448–455, 2013.

[23] Ana Carolina Lorena and André CPLF De Carvalho. Uma introdução às support vector machines. *Revista de Informática Teórica e Aplicada*, 14(2):43–67, 2007.

[24] Hervé Abdi, Dominique Valentin, and Betty Edelman. *Neural networks*. Number 124. Sage, 1999.

[25] T Soni Madhulatha. An overview on clustering methods. *arXiv preprint arXiv:1205.1117*, 2012.

[26] Carlos Oscar Sánchez Sorzano, Javier Vargas, and A Pascual Montano. A survey of dimensionality reduction techniques. *arXiv preprint arXiv:1403.2877*, 2014.

[27] Jan Philipp Albrecht. How the gdpr will change the world. *Eur. Data Prot. L. Rev.*, 2:287, 2016.

[28] Lawrence O Gostin, Laura A Levit, Sharyl J Nass, et al. Beyond the hipaa privacy rule: enhancing privacy, improving health through research. 2009.

[29] K Shailaja, Banoth Seetharamulu, and MA Jabbar. Machine learning in healthcare: A review. In *2018 Second international conference on electronics, communication and aerospace technology (ICECA)*, pages 910–914. IEEE, 2018.

[30] Irene Y Chen, Emma Pierson, Sherri Rose, Shalmali Joshi, Kadija Ferryman, and Marzyeh Ghassemi. Ethical machine learning in healthcare. *Annual review of biomedical data science*, 4:123–144, 2021.

[31] Muhammad Aurangzeb Ahmad, Carly Eckert, and Ankur Teredesai. Interpretable machine learning in healthcare. In *Proceedings of the 2018 ACM international conference on bioinformatics, computational biology, and health informatics*, pages 559–560, 2018.

[32] Matthew F Dixon, Igor Halperin, and Paul Bilokon. *Machine learning in finance*, volume 1170. Springer, 2020.

[33] Shamima Ahmed, Muneer M Alshater, Anis El Ammari, and Helmi Hammami. Artificial intelligence and machine learning in finance: A bibliometric review. *Research in International Business and Finance*, 61:101646, 2022.

[34] Hamed Ghoddusi, Germán G Creamer, and Nima Rafizadeh. Machine learning in energy economics and finance: A review. *Energy Economics*, 81:709–727, 2019.

[35] Vinicius Andrade Brei et al. Machine learning in marketing: Overview, learning strategies, applications, and future developments. *Foundations and Trends® in Marketing*, 14(3):173–236, 2020.

[36] Eric WT Ngai and Yuanyuan Wu. Machine learning in marketing: A literature review, conceptual framework, and research agenda. *Journal of Business Research*, 145:35–48, 2022.

[37] Mithun S Ullal, Iqbal Thonse Hawaldar, Rashmi Soni, and Mohammed Nadeem. The role of machine learning in digital marketing. *Sage Open*, 11(4):21582440211050394, 2021.

[38] Fotios Zantalis, Grigorios Koulouras, Sotiris Karabetsos, and Dionisis Kandris. A review of machine learning and iot in smart transportation. *Future Internet*, 11(4):94, 2019.

[39] Limon Barua, Bo Zou, and Yan Zhou. Machine learning for international freight transportation management: a comprehensive review. *Research in Transportation Business & Management*, 34:100453, 2020.

[40] Ali Tizghadam, Hamzeh Khazaei, Mohammad HY Moghaddam, Yasser Hassan, et al. Machine learning in transportation, 2019.

[41] Charles Severance and Kevin Dowd. *High performance computing*. OpenStax CNX, 2010.

[42] Eduardo Pinheiro, Ricardo Bianchini, Enrique V Carrera, and Taliver Heath. Load balancing and unbalancing for power and performance in cluster-based systems. Technical report, Rutgers University, 2001.

[43] Chnar Mustafa Mohammed and Subhi RM Zeebaree. Sufficient comparison among cloud computing services: Iaas, paas, and saas: A review. *International Journal of Science and Business*, 5(2):17–30, 2021.

[44] Somayya Madakam, Vihar Lake, Vihar Lake, Vihar Lake, et al. Internet of things (iot): A literature review. *Journal of Computer and Communications*, 3(05):164, 2015.

[45] Leslie Lamport. Paxos made simple. *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)*, pages 51–58, 2001.

[46] Diego Ongaro and John Ousterhout. The raft consensus algorithm. *Lecture Notes CS*, 190:2022, 2015.

[47] Sang H Son and Rasikan David. Design and analysis of a secure two-phase locking protocol. In *Proceedings Eighteenth Annual International Computer Software and Applications Conference (COMPSAC 94)*, pages 374–379. IEEE, 1994.

[48] Dong Young Yoon, Mosharaf Chowdhury, and Barzan Mozafari. Distributed lock management with rdma: Decentralization without starvation. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1571–1586, 2018.

[49] Seth Gilbert and Nancy Lynch. Perspectives on the cap theorem. *Computer*, 45(2):30–36, 2012.

[50] Nazrul Hoque, Dhruba K Bhattacharyya, and Jugal K Kalita. Botnet in ddos attacks: trends and challenges. *IEEE Communications Surveys & Tutorials*, 17(4):2242–2270, 2015.

[51] Pitch Patarasuk and Xin Yuan. Bandwidth optimal all-reduce algorithms for clusters of workstations. *Journal of Parallel and Distributed Computing*, 69(2):117–124, 2009.

[52] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25, 2012.

[53] Alexandros V Gerbessiotis and Leslie G Valiant. Direct bulk-synchronous parallel algorithms. *Journal of parallel and distributed computing*, 22(2):251–267, 1994.

[54] Shanshan Zhang, Ce Zhang, Zhao You, Rong Zheng, and Bo Xu. Asynchronous stochastic gradient descent for dnn training. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6660–6663. IEEE, 2013.

[55] Soummya Kar and José MF Moura. Distributed consensus algorithms in sensor networks: Quantized data and random link failures. *IEEE Transactions on Signal Processing*, 58(3):1383–1400, 2009.

[56] Dongyan Huang, Xiaoli Ma, and Shengli Zhang. Performance analysis of the raft consensus algorithm for private blockchains. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 50(1):172–181, 2019.

[57] Yuyang Deng, Mohammad Mahdi Kamani, and Mehrdad Mahdavi. Distributionally robust federated averaging. *Advances in neural information processing systems*, 33:15111–15122, 2020.

[58] Hongbo Si, Boon Loong Ng, Md Saifur Rahman, and Jianzhong Zhang. A novel and efficient vector quantization based cpri compression algorithm. *IEEE Transactions on vehicular technology*, 66(8):7061–7071, 2017.

[59] Sixin Zhang, Anna E Choromanska, and Yann LeCun. Deep learning with elastic averaging sgd. *Advances in neural information processing systems*, 28, 2015.

[60] Alysson Neves Bessani, Eduardo Pelison Alchieri, Miguel Correia, and Joni Silva Fraga. Depspace: a byzantine fault-tolerant coordination service. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, pages 163–176, 2008.

[61] Fadila Zerka, Samir Barakat, Sean Walsh, Marta Bogowicz, Ralph TH Leijenaar, Arthur Jochems, Benjamin Miraglio, David Townend, and Philippe Lambin. Systematic review of privacy-preserving distributed machine learning from federated databases in health care. *JCO clinical cancer informatics*, 4:184–200, 2020.

[62] Usha Manasi Mohapatra, Babita Majhi, and Suresh Chandra Satapathy. Financial time series prediction using distributed machine learning techniques. *Neural Computing and Applications*, 31:3369–3384, 2019.

[63] Chao Yu, Xin Wang, Xin Xu, Minjie Zhang, Hongwei Ge, Jiankang Ren, Liang Sun, Bingcai Chen, and Guozhen Tan. Distributed multiagent coordinated learning for autonomous driving in highways based on dynamic coordination graphs. *Ieee transactions on intelligent transportation systems*, 21(2):735–748, 2019.

[64] Mohamed Amine Ferrag, Lei Shu, Hamouda Djallel, and Kim-Kwang Raymond Choo. Deep learning-based intrusion detection for distributed denial of service attack in agriculture 4.0. *Electronics*, 10(11):1257, 2021.

[65] TensorFlow Developers. Tensorflow. *Zenodo*, 2022.

[66] Chuck Lam. *Hadoop in action*. Simon and Schuster, 2010.

[67] Sumit Mund. *Microsoft azure machine learning*. Packt Publishing Ltd, 2015.

[68] Edo Liberty, Zohar Karnin, Bing Xiang, Laurence Rouesnel, Baris Coskun, Ramesh Nallapati, Julio Delgado, Amir Sadoughi, Yury Astashonok, Piali Das, et al. Elastic machine learning algorithms in amazon sagemaker. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 731–737, 2020.

[69] Erin LeDell and Sebastien Poirier. H2o automl: Scalable automatic machine learning. In *Proceedings of the AutoML Workshop at ICML*, volume 2020. ICML, 2020.

[70] Nguyen Van Thieu. Permetrics: A framework of performance metrics for artificial intelligence models, July 2020.

[71] Robert E Tillman. Structure learning with independent non-identically distributed data. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1041–1048, 2009.

[72] Chenghao Liu, Steven CH Hoi, Peilin Zhao, and Jianling Sun. Online arima algorithms for time series prediction. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[73] Sima Siami-Namini, Neda Tavakoli, and Akbar Siami Namin. A comparison of arima and lstm in forecasting time series. In *2018 17th IEEE international conference on machine learning and applications (ICMLA)*, pages 1394–1401. IEEE, 2018.

[74] Plamen P Angelov, Eduardo A Soares, Richard Jiang, Nicholas I Arnold, and Peter M Atkinson. Explainable artificial intelligence: an analytical review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 11(5):e1424, 2021.

[75] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " 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*, pages 1135–1144, 2016.

[76] Mukund Sundararajan and Amir Najmi. The many shapley values for model explanation. In *International conference on machine learning*, pages 9269–9278. PMLR, 2020.

Appendix A

HDFS

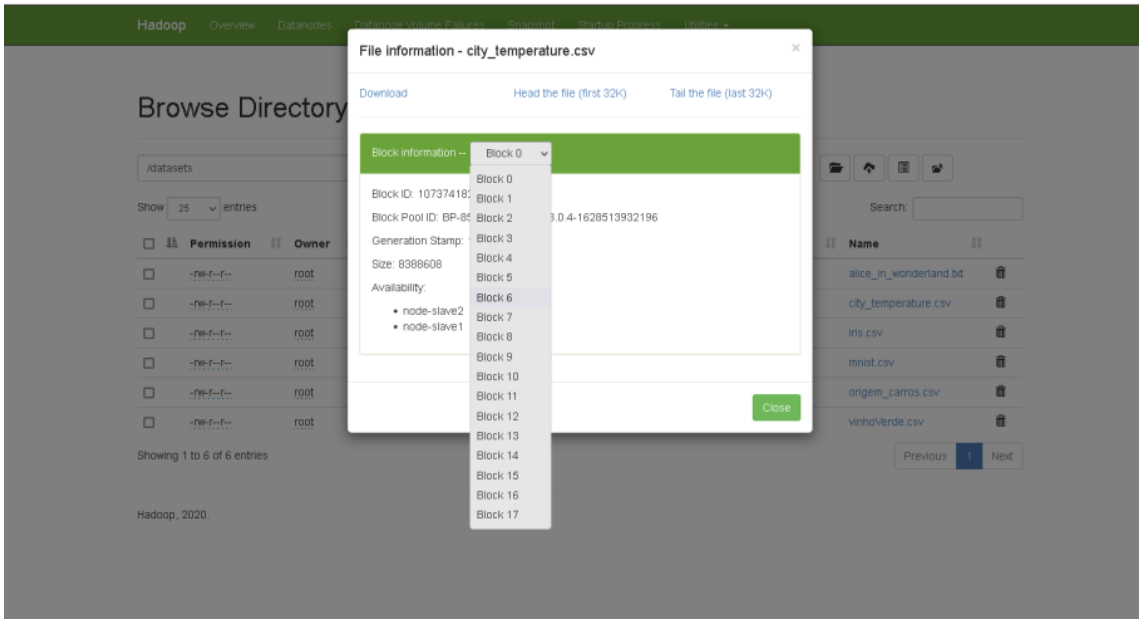


Figure 17: Dataset block division in HDFS.

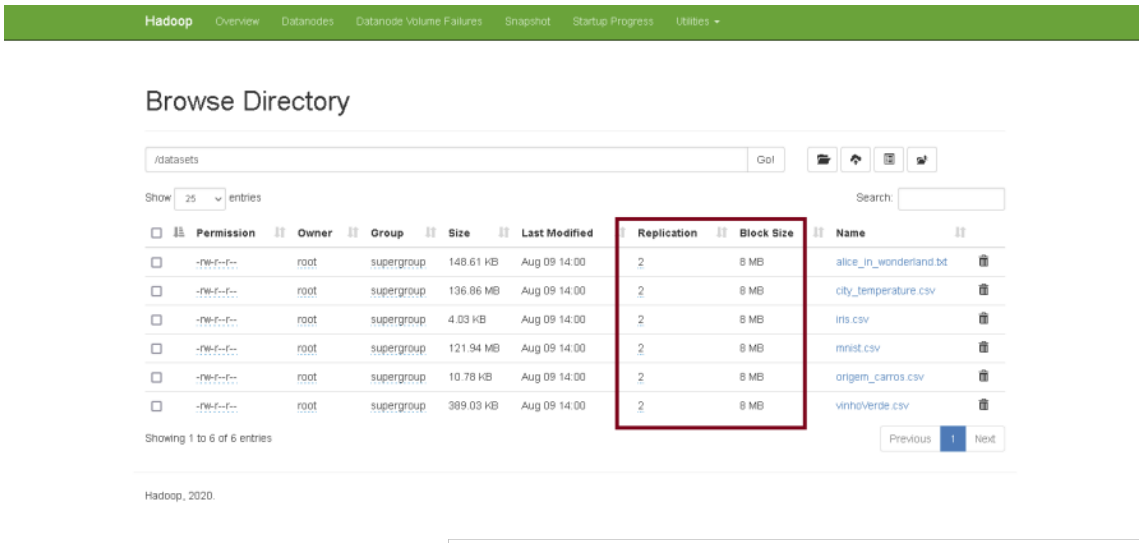


Figure 18: HDFS block size and replicator factor.

Appendix B

Frontend

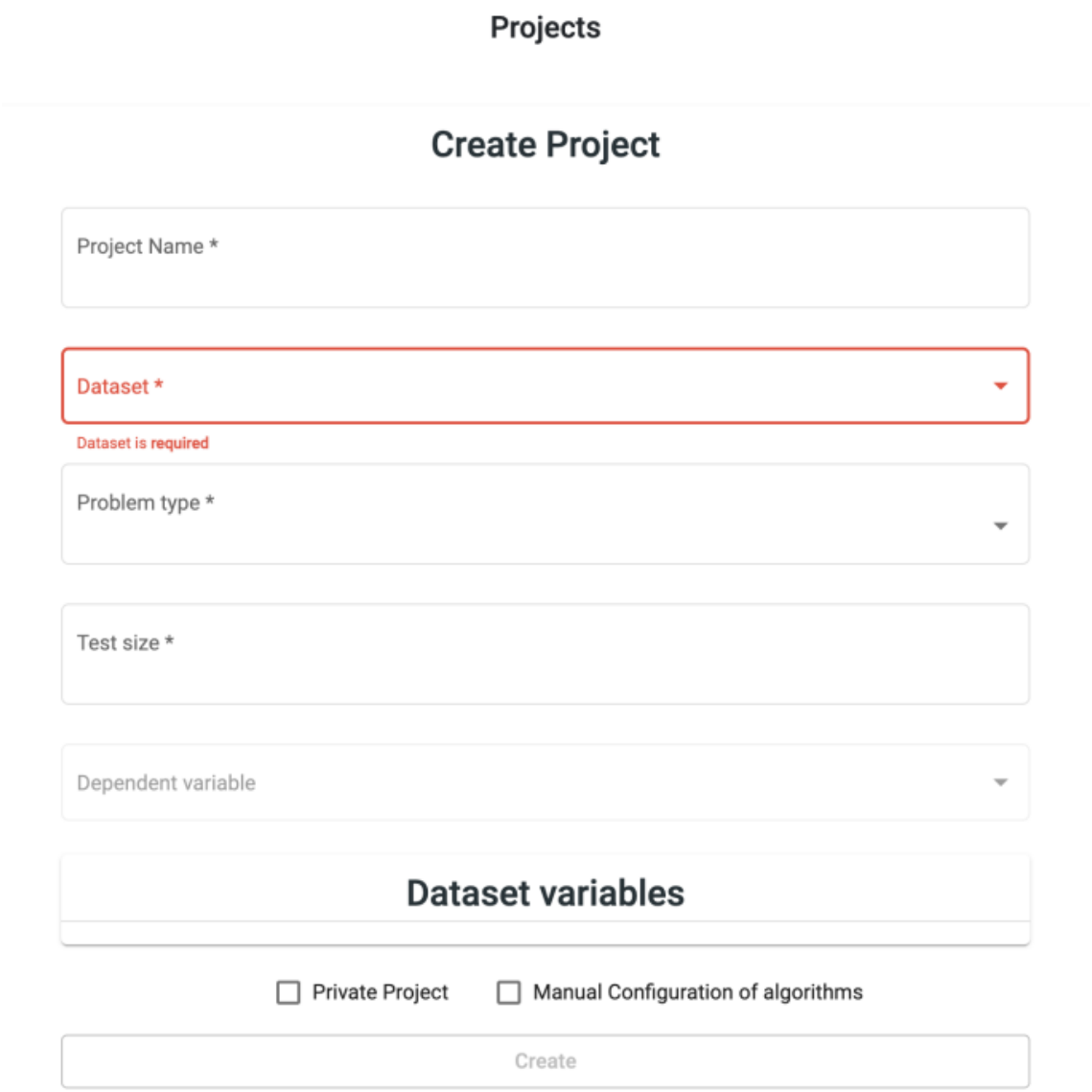


Figure 19: Project creation form.

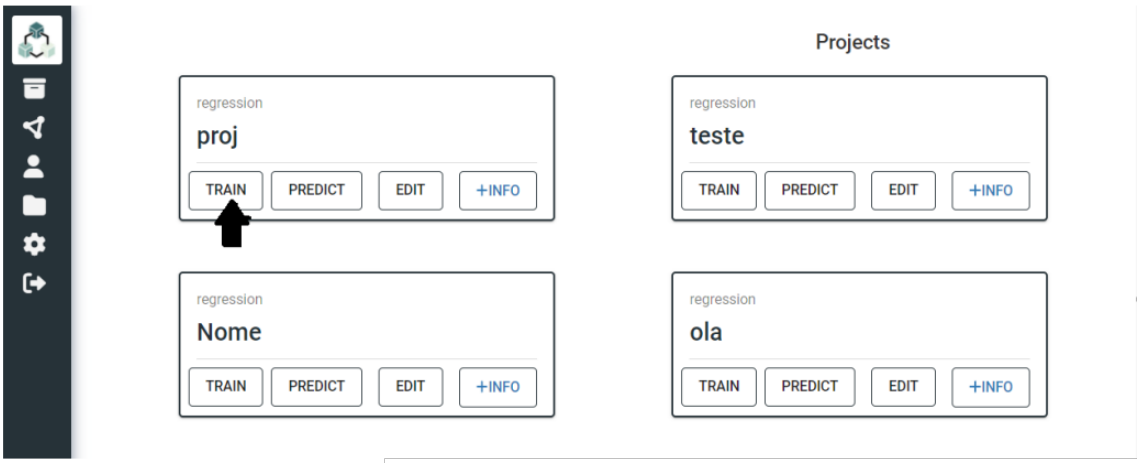


Figure 20: List of projects and its functionalities.

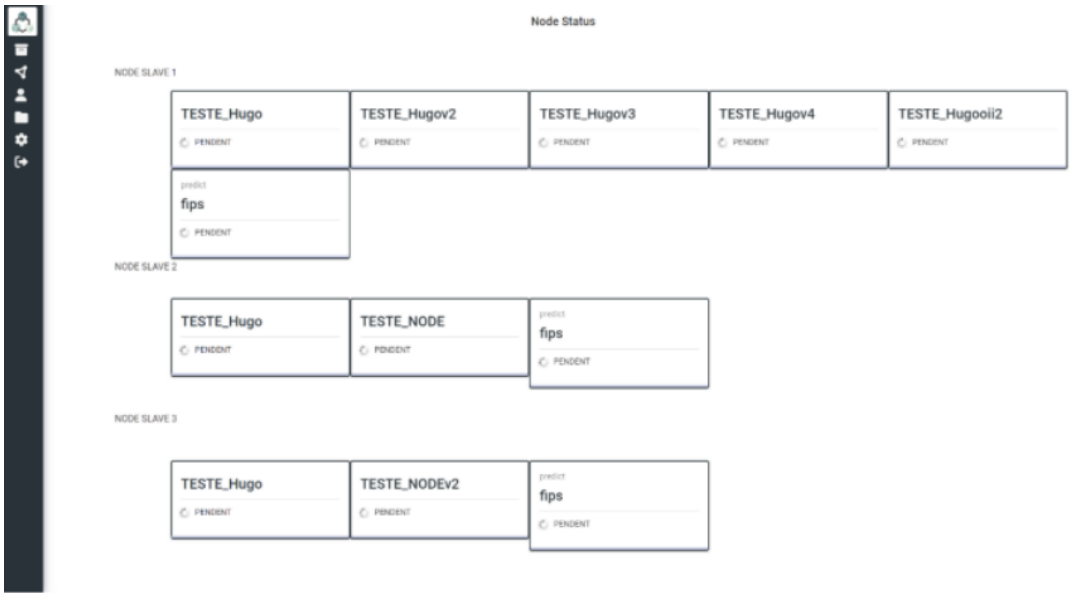


Figure 21: System worker nodes wait list.