Prediction of ML Operator Performance

Angeliki Emmanouela Syrri
ECE
NTUA
Athens, Greece
Email: el18811@mail.ntua.gr

Christina Diamanti
ECE
NTUA
Athens, Greece
Email: el18203@mail.ntua.gr

Emmanouil Almpantakis
ECE
NTUA
Athens, Greece
Email: el15129@mail.ntua.gr

***Abstract*—In this research paper, we conduct an investigation into forecasting the effectiveness of machine learning operations within the Apache Spark framework. Our goal is to enhance the efficiency of big data analytics. To accomplish this, we established a multi-node Spark cluster to handle substantial datasets using PySpark MLlib's machine learning algorithms. Our primary emphasis was placed on conducting thorough experiments involving three specific operators: k-means clustering, Random Forest Regression, and Word2Vec. Based on the data and outcomes we gathered, we developed models capable of reliably predicting how each operator performs under various dataset conditions.**

I. Introduction

In today's world of vast data, organizations increasingly turn to advanced analytics to unlock valuable insights from complex datasets. Among the tools available, Apache Spark has gained popularity for its ability to process large data and support various machine learning methods. However, running machine learning tasks in Apache Spark can be intricate and time-consuming, potentially hindering the efficiency of data analytics.

To address this challenge, this paper introduces an innovative approach to enhance the performance of machine learning tasks in Apache Spark. This approach involves using machine learning techniques to predict how these tasks will perform without actually running them. We collect multiple performance measurements over time and employ a learning algorithm to create a model that can accurately forecast performance.

The primary objective of this study is to provide a practical solution for improving the efficiency of data analytics by predicting how machine learning tasks will perform in Apache Spark. In the following sections, we outline our data collection and processing methodology, describe the machine learning algorithms we use to build performance prediction models, and present the evaluation of our proposed approach. The results of this study can assist organizations in optimizing the performance of machine learning tasks in Apache Spark, enabling them to extract insights from large and complex datasets more efficiently.

The code for the project can be found in the following GitHub link: GitHub Repo (https://github.com/manosalmpa/cs_da.git)

II. Infrastructure and System Configuration

Within this section, we describe the setup of a multi-node Apache Spark cluster (comprising one master and one slave) specifically designed to handle large datasets through collaborative processing.

A. Installation Steps

The cluster we created was deployed on Virtual Machines (VMs) hosted within the Okeanos service. Accessing the machines was accomplished through the utilization of the SSH protocol. Both systems operate using Ubuntu 16 LTS. While each VM is initially assigned an IPv6 address upon creation, we also allocated a public IPv4 address exclusively to the master VM. Communication between these machines takes place via a dedicated private network.

1) Virtual Machine Setup: Access the Okeanos-Knossos interface and provision a Virtual Machine (VM) for your cluster. Assign a public IP address to the VM to facilitate external access and management.

2) SSH Connection: Establish an SSH connection to the VM using the provided command. Replace 'user' with your username and 'snf-33040.ok-kno.grnetcloud.net' with your VM's hostname. Enter your password as prompted.

3) Python 3.8 Installation:

- Update package lists using the command sudo apt update.
- Install essential dependencies by running sudo apt install with a list of required packages.
- Download Python 3.8 source code from the official Python website.
- Extract and navigate to the Python source directory.
- Configure and build Python 3.8 as specified.
- Verify Python 3.8 installation by checking its version.
- Remove old Python links to avoid conflicts.

4) Pip Installation:

- Return to the home directory.
- Download the get-pip.py script.
- Install pip using the newly installed Python 3.8.

5) PySpark Installation: Install PySpark version 3.1.3 using pip.

6) Apache Spark Installation:

- Download Apache Spark version 3.1.3 from the official website.
- Extract the downloaded archive to a specified directory.

- Configure environment variables by adding relevant lines to the .bashrc file.
- Source the updated .bashrc to apply the changes.

7) Java Installation:

- Install Java Development Kit (JDK) 8 on your VM.
- Verify the Java installation by checking its version.

8) Cluster Setup (1 Master and 1 Worker):

- Create a network within the Okeanos environment and assign IP addresses to each VM.
- Navigate to the Spark configuration directory.
- Create a file named spark-env.sh.
- Edit spark-env.sh and set the SPARK MASTER HOST to the appropriate IP address.
- Start the Spark Master.
- Deploy worker nodes as needed following specified custom resource configurations.

9) Expanding the Cluster:

- For each additional VM, repeat Steps 3, 6, and 7 to expand the cluster.

10) Accessing the Spark Cluster:

- To access the Spark cluster, you can use a web browser to view the Spark Web UI. By default, you can access it locally on the master node using the following URL: http://localhost:8080/ This will provide you with an overview of the Spark cluster's status, including information about running applications, workers, and more.

B. System and Software Description

In this section, we present an overview of the system and software components utilized in our research infrastructure. The system is designed to support large-scale data processing and analysis tasks, and it leverages a combination of virtual machines (VMs), Apache Spark, PySpark, and related libraries. Below, we provide a comprehensive description of each component:

1) Virtual Machines (VMs): The core of our research infrastructure consists of virtual machines hosted within the Okeanos-Knossos environment. VMs offer a flexible and scalable approach to resource allocation, enabling us to create a distributed computing environment for data processing tasks. Specifically:

- **VM Setup:** VMs were provisioned using the Okeanos-Knossos interface, and each VM was assigned a public IP address to facilitate external access and management.
- **SSH Connectivity:** Secure Shell (SSH) connections were established to the VMs, allowing remote access and administration.

2) Apache Spark: Apache Spark is a prominent distributed data processing framework that plays a central role in our infrastructure. Key features and components of Apache Spark include:

- **Installation:** We installed Apache Spark version 3.1.3, which provides high-performance, distributed data processing capabilities.
- **Cluster Setup:** A multi-node Spark cluster was configured, consisting of a master node and one or more worker nodes. The Spark cluster is responsible for parallelized data processing, enabling efficient computation across distributed resources.
- **Web User Interface (UI):** The Spark Web UI, accessible through a web browser, serves as the primary interface for monitoring and managing the cluster. It provides real-time information on the cluster's status, running applications, and resource utilization.

3) PySpark: PySpark, the Python API for Apache Spark, is instrumental in enabling machine learning and data analysis tasks on the Spark cluster. Key aspects of PySpark include:

- **Installation:** PySpark version 3.1.3 was installed using the Python package manager, pip. PySpark enhances the accessibility and ease of development of Spark-based applications.
- **Machine Learning:** PySpark offers extensive support for machine learning tasks through its MLlib library, which includes a wide range of pre-implemented machine learning algorithms.

4) Java Development Kit (JDK): The Java Development Kit (JDK) version 8 was installed on the VMs to support the execution of Spark applications. This JDK version is compatible with Apache Spark 3.1.3 and ensures that Java-based components of Spark function correctly.

5) Accessing the Spark Cluster: Accessing the Spark cluster is facilitated through a web browser. Users can access the Spark Web UI either locally or remotely, depending on their monitoring and management requirements:

- **Local Access:** The Spark Web UI can be accessed locally on the master node using the URL `http://localhost:8080/`. It provides detailed information about the cluster's performance and status.
- **Remote Access:** To access the Spark Web UI remotely, users can replace "localhost" in the URL with the public IP address of the master node. This enables remote monitoring and management of the Spark cluster from external locations.

In summary, our research infrastructure is built upon virtual machines and Apache Spark, enabling large-scale data processing and analysis tasks. PySpark and related libraries enhance the capabilities of the Spark cluster, while the Spark Web UI provides a user-friendly interface for cluster monitoring and management. This system and software configuration empower researchers to efficiently process and analyze extensive datasets.

III. Experiment Stage and The Prediction Model

A. Experiment Structure

In this section, we describe the methodology employed to address the problem of modeling the performance of machine learning operators executed in big data analytics runtimes. Our approach involves the installation and setup of a specific runtime, selection of operators for modeling, data generation

and loading, measurement of performance, and the modeling step. The goal is to create a prediction model for operator performance without executing them.

1) Installation and Setup of Runtime Environment: The first phase of our experiment focuses on the installation and setup of an open-source, distributed analytics engine. We opted for Apache Spark as our chosen runtime due to its widespread use and versatility. The following steps were undertaken:

1) **Resource Selection**: We utilized local and Okeanos-based resources for setting up the Spark runtime environment. These resources were carefully configured to ensure compatibility and performance.
2) **Apache Spark Installation**: We installed Apache Spark, a distributed analytics engine, on the selected resources. This step included downloading the latest version, configuring system variables, and setting up the necessary dependencies.

2) Selection of Operators for Modeling: To build our prediction model, we selected a minimum of three machine-learning operators that would serve as the focal points of our study. Our choice of operators aimed to achieve diversity while maintaining a common theme within the selected family (in this case, machine learning operators). The operators we use in our project are:

1) **k-means:**: This operator is known for its effectiveness in clustering data points into groups based on similarity. It's particularly useful when we want to uncover underlying patterns or groupings within our dataset.
2) **Random Forest Regression**: Random Forest is a versatile ensemble learning method that is widely used for both classification and regression tasks. It combines the power of multiple decision trees to provide accurate predictions, making it a robust choice for various modeling scenarios.
3) **Word2Vec**: Unlike traditional machine learning operators, Word2Vec is more specialized for natural language processing tasks. It is used for word embedding, which means it transforms words or phrases into numerical vectors, making it easier for machine learning models to process and understand textual data.

3) Data Generation and Loading: In this phase, we focused on creating representative sample input data for the selected operators. Our approach involved using an artificial data generator to simulate various data scenarios. Key considerations for this step were:

1) **Data Variety**: We generated data of different sizes, ranging from small/medium to quite large datasets that would not fit in main memory.
2) **Data Structures**: We varied the data structures, including different types of graphs and data points with varying dimensions. This diversity allowed us to test the operators under different conditions.

4) Measurement of Performance and Modeling Step: This section outlines the core of our experiment, which includes the measurement of operator performance and the modeling step. The following key activities were performed:

1) **Performance Measurement**: For each combination of data and operator, we executed the operator and recorded performance metrics. These metrics included total running time, min/max/avg CPU usage, main memory cluster usage, and other relevant statistics.
2) **Data Collection**: The collected performance data from various experiments were organized and prepared for the modeling step.
3) **Modeling**: We employed machine learning techniques to create prediction models. A variety of learners, including neural networks, regression models, and random forests, were considered. The goal was to develop accurate prediction models that could estimate an operator's behavior, such as its running time and memory requirements, with minimal error when applied to unseen data inputs.

B. Experiment Methodology

1) Initialization of Spark Cluster: To facilitate our experiments and ensure consistent and controlled execution environments, we initialized our Spark cluster with the following configurations: Here, we set up the Spark cluster with specified memory allocations for executors and drivers, as well as the number of cores to be utilized. The memory fraction and periodic garbage collection intervals were carefully tuned to optimize cluster performance for our experiments.

More about our setup decisions:

- **Cluster Master Configuration:**
  1) **Configuration Decision**: We specified the cluster master with the address "spark://10.0.0.1:7077".
  2) **Rationale**: This configuration identifies the master node's location, allowing the Spark cluster to establish proper communication.
- **Executor Memory Allocation:**
  1) **Configuration Decision**: We allocated 2 gigabytes of memory per executor using the configuration spark.executor.memory.
  2) **Rationale**: Allocating a specific amount of memory to each executor helps manage resource utilization efficiently. The chosen value of 2GB was based on resource availability and task requirements.
- **Executor Cores:**
  1) **Configuration Decision**: We assigned 2 CPU cores per executor with spark.executor.cores.
  2) **Rationale**: Allocating CPU cores per executor optimizes parallel processing. The choice of 2 cores was made to balance parallelism with resource availability.
- **Driver Memory Allocation:**
  1) **Configuration Decision**: We allocated 2 gigabytes of memory to the driver node using spark.driver.memory.
  2) **Rationale**: The driver node is responsible for coordinating tasks and collecting results. Allocating

sufficient memory ensures it can handle these responsibilities effectively.

Memory Fraction:

1) **Configuration Decision**: We set the memory fraction to 0.6 with spark.memory.fraction.
2) **Rationale**: This configuration determines the portion of memory allocated for execution and storage. A value of 0.6 was chosen to provide an appropriate balance between computation and caching.

Periodic Garbage Collection Interval:

1) **Configuration Decision**: We configured periodic garbage collection intervals to occur every minute with spark.cleaner.periodicGC.interval.
2) **Rationale**: Frequent garbage collection helps reclaim memory and maintain cluster stability. A one-minute interval was chosen to manage memory efficiently.

Each configuration decision was made with careful consideration of resource availability, cluster stability, and task requirements. These settings collectively aim to provide an optimal environment for executing machine learning operators efficiently and effectively within the Spark cluster.

2) Data Generation: Generating appropriate datasets for our experiments was a critical component of our research. Several factors guided our decision to create custom datasets:

1) **Focus on Performance Evaluation**: Our primary objective was to evaluate the execution time and resource consumption of machine learning operators. Therefore, the correctness and realism of the datasets were of secondary importance.
2) **Dynamic Dataset Creation**: We needed the flexibility to generate datasets with varying dimensionalities and sizes on demand. This dynamic capability was essential for comprehensive experimentation.
3) **Efficiency and Control**: Gathering diverse datasets from external sources would have been time-consuming and impractical. Creating our own datasets allowed us to maintain control and efficiency in the experimentation process.

To achieve this, we developed customized data generators tailored to each operator under evaluation. These generators accepted parameters such as the number of samples and features to produce the required datasets. The generated datasets were stored on disk in specific files, accompanied by metadata describing attributes such as the dataset's dimensions, clusters (where applicable), and size in megabytes.

During our experiments, we discovered that leveraging functions from the sklearn.datasets package, specifically $make\_blobs$ and $make\_regression$, proved highly effective for generating datasets. Additionally, we utilized the $dump\_svmlight\_file$ function from the same package to save the datasets in the LibSVM format, which was suitable for the K-Means and RandomForest algorithms. To further enhance efficiency, we incorporated batching techniques into our data generation scripts for creating large datasets without encountering performance issues.

For the Word2Vec operator, we designed a custom data generator adhering to the same principles. However, it did not rely on external data generation tools. Instead, we generated datasets in the form of sentences, with each sentence occupying a single row in the dataset. This approach was chosen because the Word2Vec algorithm requires specific data structures. Sentences were constructed by randomly selecting english words downloaded from NLTK resource. NLTK (Natural Language Toolkit) is a leading platform for building Python programs to work with human language data. . It offers user-friendly interfaces for more than 50 datasets and lexical references. Importantly, we maintained consistent dimensions across all phrases to ensure a uniform dimensional structure across various tests.

In summary, our custom dataset generation approach provided us with precise control over experimentation conditions, enabling comprehensive testing of machine learning algorithms across a spectrum of scenarios. This level of control was pivotal in advancing our research objectives.

C. Experiment Execution

During the execution phase, we employed models from the PySpark library, adhering to default parameter settings for the most part. However, we made an exception for the k-means model, where we dynamically adjusted the number of centers. Following this, we applied each model to our experimental dataset. It's important to emphasize that we refrained from carrying out predictions or scoring, as these tasks were outside the scope of our study. Throughout the majority of our experiments, we vigilantly monitored the CPU and memory usage of the Spark clusters and VMs. Our observations consistently pointed to memory as the primary limiting factor in the system. To tackle this issue, we consistently cleared all variables and unpersisted datasets, effectively reducing the memory burden.

In the execution phase of our machine learning operator, we follow a systematic approach. To begin, we establish a Spark session using the create_spark_session function. The Spark session configuration is crucial a crucial step for managing tasks related to Spark. We dynamically determine the dimensions of the dataset that our machine learning operator will work with. This includes randomly selecting the number of samples (num_samples) and the number of features (num_features) within predefined ranges. We then load the dataset and configure the parameters and settings of the machine learning model. We employ a monitoring function to oversee the training process of the machine learning model. This function logs relevant training results, which include the model's performance metrics. Finally, we ececute resource

cleanup to manage resources efficiently and terminate the spark session releasing all associated resources. Proper session termination is essential to ensure that resources are not consumed unnecessarily.

D. Experiment Results Visualization

To enhance our comprehension of the datasets and improve the efficiency of our model training, we performed an extensive examination of the data and generated graphical representations in order to be able to visualize patterns in the generated data.

1) Kmeans Results: The plot illustrating "Memory usage" against "Dataset size" reveals an almost flawless linear connection between the dataset's size and memory usage.

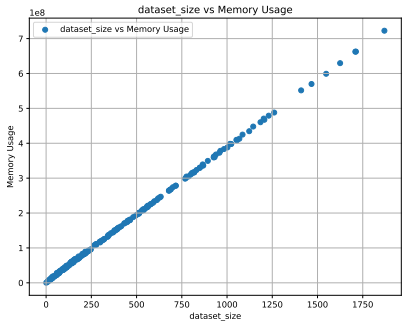

Fig. 1. Dataset Size and Memory Usage Diagram

The plot showing "Total time" versus "Dataset size" clearly indicates a nearly straight-line connection between these factors. This emphasizes the importance of keeping this column intact during data preprocessing.

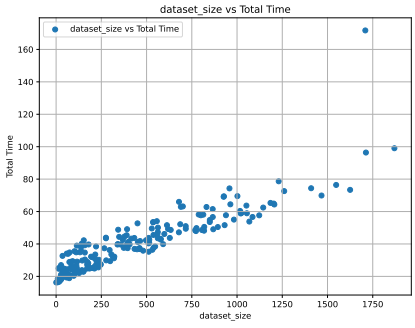

Fig. 2. Dataset Size and Total Time Diagram

The relationship between the number of features and memory usage distribution is more intricate, showing a quicker increase in memory usage as the number of samples or features rises. This results in a broader range of memory usage values for larger datasets in comparison to smaller ones, which poses a greater challenge in accurately predicting the memory usage needed for a specific task.

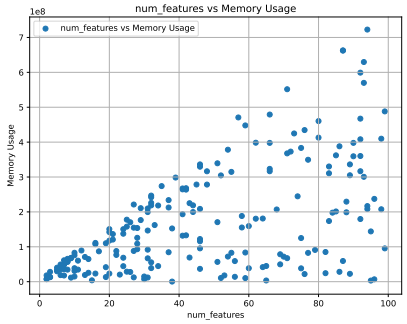

Fig. 3. Number of Features and Memory Usage Diagram

As far as the relationship between the number of features and the total time results is concerned the visualizations clearly indicate that the data is evenly spread and doesn't contain many unusual data points, which makes it well-suited for training a predictive model.

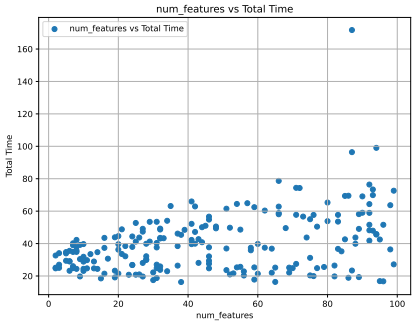

Fig. 4. Number of Features and Total Time Diagram

The Memory Usage to Number of samples distribution shows an expanding upper limit as the number of samples or features increases.

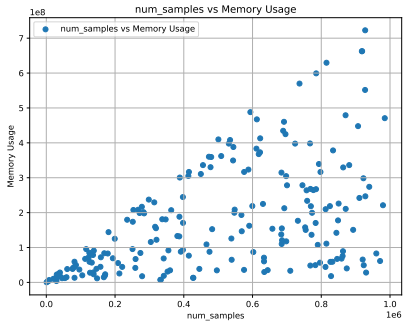

Fig. 5. Number of Samples and Memory Usage Diagram

The relationships between the number of features and number of samples display a linearly increasing upper boundary and a steady lower boundary. The presence of a linearly growing upper boundary suggests that as the number of samples or

features grows, the training time also increases proportionally. This characteristic is desirable because it indicates that the model can effectively handle larger datasets as they grow in size.

Furthermore, the consistent lower boundary of the distributions indicates that even with smaller datasets, the training time remains sufficiently substantial to accurately train the model. This is crucial for achieving accurate predictions, even when working with limited data.

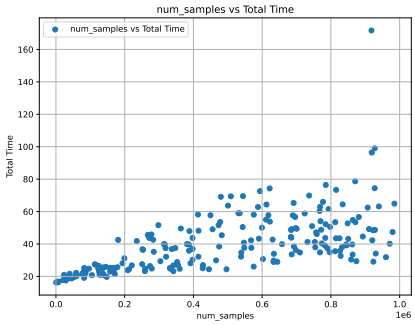

Fig. 6. Number of Samples and Total Time Diagram

2) Random Forest Results: The connection between Total Time and Dataset Size and Memory Usage and Dataset Size is not linear but it is discernible once more.

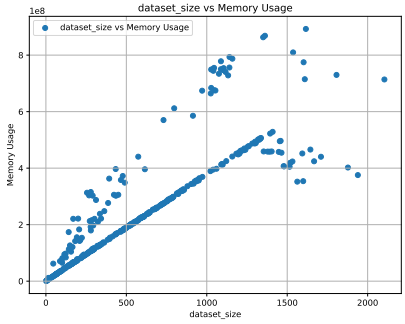

Fig. 7. Dataset Size and Memory Usage Diagram

Although there appears to be a linear correlation, there is a considerable number of data points that exhibit variations.

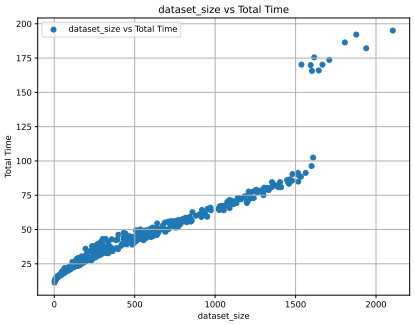

Fig. 8. Dataset Size and Total Time Diagram

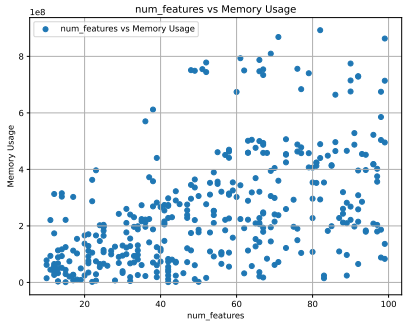

Fig. 9. Number of Features and Memory Usage Diagram

The Total Time to Number of features distribution shows an expanding upper limit as the number of samples or features increases.

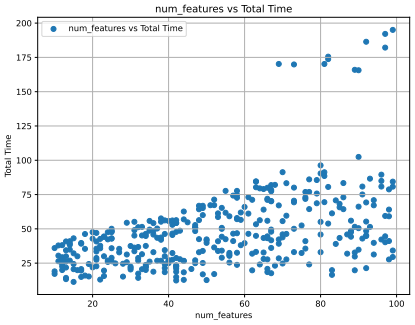

Fig. 10. Number of Features and Total Time Diagram

In the relationship between both the number of samples and the number of features with memory usage we can observe the presence of some outliers. These irregularities in the dataset do pose a challenge but it is achievable to construct a precise and trustworthy machine learning model with thoughtful choice of suitable methods and algorithms. The diagrams also exhibit a ”growing upper limit” pattern, indicating an increase in dataset size and complexity.

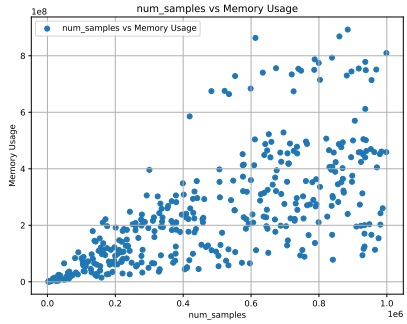

Fig. 11. Number of Samples and Memory Usage Diagram

In the Number of Samples to Total Time diagram we can observe, once more, the existence of a steadily increasing upper limit.

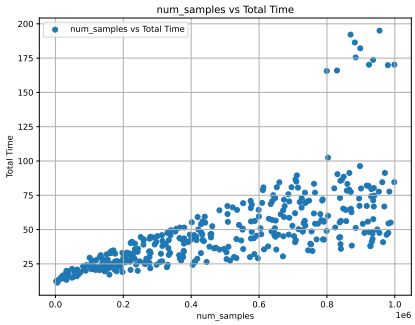

Fig. 12. Number of Samples and Total Time Diagram

3) Word2Vec Results: We can see that the dataset corresponding to the word2vec operator differs dramatically from the other two datasets. One of the most noticeable distinctions is that the values on the Time Taken and Memory Usage axes will be significantly higher in comparison. This distinction might be traced to the fundamental nature of the word2vec algorithm, which works with vast amounts of text data and includes sophisticated computations.

The "Memory Usage and Dataset Size Relationship" is not linear, as in many other visualizations that will follow. While not uniform or random, there is a distinct variance in Memory utilization statistics, even for datasets with extremely comparable Dataset size properties.

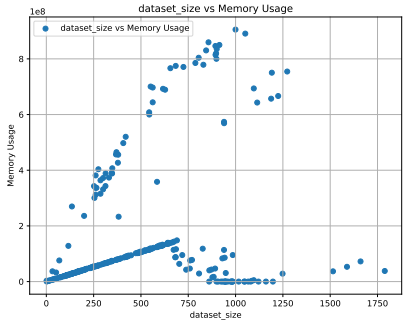

Fig. 13. Dataset Size and Memory Usage Diagram

In the diagram below, we can observe the linear relationship between the dataset size to total time with only very few outliers.

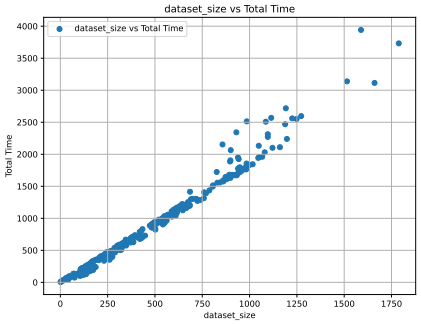

Fig. 14. Dataset Size and Total Time Diagram

As the Number of Features increases, Memory Usage has an upper bound excluding some outliers.

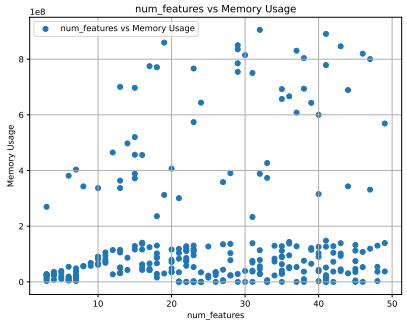

Fig. 15. Number of Features and Memory Usage Diagram

The total Time to Number of features distribution shows an expanding upper limit as the number of samples or features increases. The total time required does not necessarily follow the same path as the number of characteristics grows.

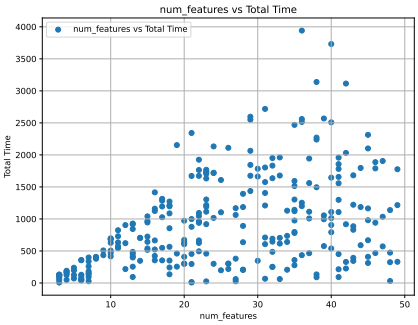

Fig. 16. Number of Features and Total Time Diagram

Given the findings obtained from the word2vec dataset regarding Memory usage, generating a model that can accurately predict the memory that the word2vec operator is going to use, is likely to be a difficult task, even with proper data preprocessing and extensive experimentation with different model architectures and hyperparameters.

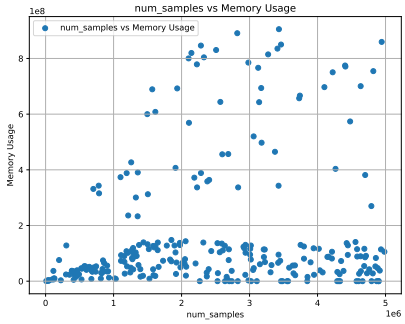

Fig. 17. Number of Samples and Memory Usage Diagram

The correlations between total time and sample number show a linearly growing upper barrier and a constant bottom boundary. The presence of a linearly increasing upper barrier implies that as the number of samples or features increases, so does the training time.

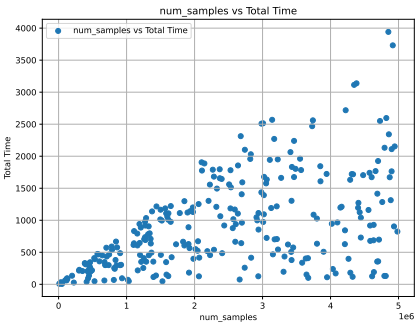

Fig. 18. Number of Samples and Total Time Diagram

This problem, namely the difficulty of creating an accurate predictor for the memory usage of word2vec, can be related to the word2vec algorithm's fundamental nature, which incorporates a substantial level of unpredictability in its computations. This randomness can lead to data inconsistencies, making it difficult to construct a strong and reliable model that can account for these differences.

As a result, while forecasting Memory Usage for the word2vec operator, it may be required to tolerate a certain level of uncertainty and imprecision.

E. The Prediction Model

1) Creating the training datasets: In order to create the training dataset for our experiment, we incorporated a monitoring mechanism within our code to capture crucial performance metrics of the Spark operators used. These metrics are pivotal in understanding the behavior and efficiency of these operators. We monitor these key metrics:

1) **Total Execution Time**: We recorded the total time taken for the execution of the Spark operator, from start to finish. This metric provides an overview of the operator's computational efficiency.
2) **CPU Usage**: To gauge the CPU utilization during the operator's execution, we collected data on the minimum, maximum, and average CPU usage across all executors. This information helps us understand the computational load imposed by the operator.
3) **Memory Usage**: Memory consumption is a critical factor in Spark applications. We measured the minimum, maximum, and average memory usage to assess how efficiently memory resources were utilized during operator execution.

To obtain this performance data, we utilized the Spark UI's REST API. We accessed the REST API of the Spark UI, which was running at a specific address (http://10.0.0.1:4040). By querying this API, we retrieved details about the executors that were actively involved in the execution of our Spark application. We extracted relevant information from these executors, including CPU usage, memory consumption, and execution duration.

Based on the experiment results we concluded that the most indicative parameters characterizing each operator's performance where training duration and memory consumption metrics. These metrics were selected as the focal points of our ultimate predction model due to their direct influence on the efficiency and effectiveness of each operator. Leveraging these statistical indicators, our objective is to precisely predict the operator's behavior and enhance the execution efficiency of the Spark application.

Once we gathered these performance metrics, we integrated them into our training dataset creation process. We read the metadata associated with our experiment from the

"data.libsvm.meta" file. Then, we wrote the collected performance statistics, including total execution time and memory usage, into our training dataset file.

2) The Datasets: The experiment results have been organized into separate datasets for each operator, which will serve as the foundational data for training the prediction models in the final phase of our study.

1) **Total Time**: the duration taken by the operator to fit the data.
2) **Memory Usage**: the amount of memory consumed by the cluster.
3) **Number of samples**: the quantity of samples contained within the dataset.
4) **Number of features**: the number of features found within each sample.
5) **Dataset size**: the size of the dataset in megabytes.

The dataset containing the results of the experiments using the kmeans operator included a seventh column representing the number of clusters within the dataset.

3) Performance Prediction Methodology: We will now describe the process of predicting the performance of the each Spark operator using various regression models. To facilitate this, we created pipelines for each regression model, allowing us to systematically preprocess and model the data.
The pipeline includes the following steps:

1) **Imputation**: Missing values are handled using the median strategy.
2) **Scaling**:Data is scaled using Min-Max scaling to ensure uniform feature scaling.
3) **Feature Selection**: VarianceThreshold is applied to remove low-variance features, which may not contribute significantly to the model.
4) **Modeling**: The Regression model is utilized within a MultiOutputRegressor, enabling the prediction of multiple output variables simultaneously.

To further optimize the performance of these models, we employed a parameter grid search approach. This involved using the GridSearchCV function from sklearn.model_selection to systematically search for the best hyperparameters for each model. Each model had its own set of parameters to fine-tune, such as fit_intercept, normalize, alpha, n_neighbors, n_estimators, and many others, depending on the specific regression algorithm. By systematically exploring and tuning these hyperparameters for each model, we aimed to identify the optimal configurations that would yield the most accurate predictions for each operator's memory and time performance.

4) Comparative Analysis of Model Performance: Within the context of our research paper, we have employed a histogram to present a comparative analysis of various machine learning regression algorithms. We assessed the effectiveness of the different models through the examination of metrics, including Mean Absolute Error (MAE) and R-squared.
The R2 score, often referred to as the coefficient of determination, is a statistical measure used to assess the goodness-of-fit of a regression model. Its range spans from 0 to 1, where a perfect fit is denoted by an R2 score of 1 and a score of 0 suggests that the model does not explain any variability, performing no better than a horizontal line.
The Mean Absolute Error is a statistical measure used to assess the average magnitude of errors in a regression model's predictions. It quantifies the absolute difference between predicted values and actual values. Lower MAE values indicate more accurate predictions, while higher values suggest larger prediction errors.
Below we mention the Regression Algorithms under investigation:

1) **RFR (Random Forest Regressor)**:The Random Forest Regressor is an ensemble learning method renowned for its capacity to generate highly accurate predictions by aggregating the outputs of multiple decision trees. It excels in handling intricate data patterns and is recognized for its robustness.
2) **LR (Linear Regression)**:Linear Regression is a fundamental and interpretable model that establishes a linear association between input features and the target variable. Its simplicity and transparency make it a common choice in regression analyses.
3) **Lasso (Lasso Regression)**:Lasso Regression, employing L1 regularization, serves as an effective tool for feature selection. It encourages sparsity in model coefficients, promoting the identification of essential predictors in high-dimensional datasets.
4) **KNN (k-Nearest Neighbors Regression)**:k-Nearest Neighbors Regression is a non-parametric algorithm that predicts values based on the consensus of its closest neighboring data points in feature space. Its adaptability renders it suitable for both regression and classification tasks.
5) **EN (Elastic Net Regression)**:Elastic Net Regression strikes a balance between feature selection and model flexibility by combining L1 (Lasso) and L2 (Ridge) regularization techniques. It is particularly advantageous when dealing with datasets containing numerous features with varying degrees of relevance.
6) **GBR (Gradient Boosting Regressor)**:The Gradient Boosting Regressor, an ensemble method, constructs decision trees sequentially, with each tree correcting the errors of its predecessor. This approach empowers the model to capture intricate data relationships and has earned it a reputation for exceptional predictive prowess.

These algorithms represent a diverse set of regression techniques commonly employed in machine learning applications.

5) Kmeans Regression: As shown in the relative histogram when it comes to memory usage, all the evaluated models perform similarly, with no significant differences.

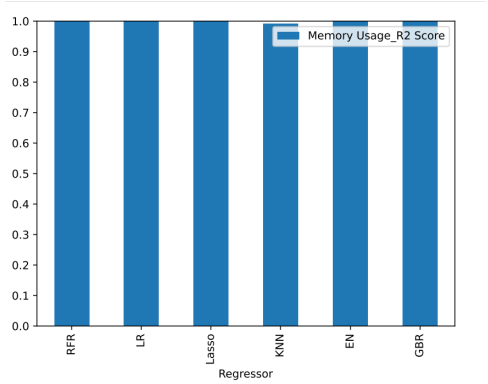

Fig. 19. Kmeans Regression - Memory Usage Performance - R2 Score

Based on the MAE values in the histogram, it appears that Linear Regression, Lasso, and Elastic Net models provide the most accurate predictions with a MAE of 1, followed by the Random Forest Regressor and Gradient Boosting Regressor with a MAE of 3. KNN, with a MAE of 8, exhibits relatively higher prediction errors on average.

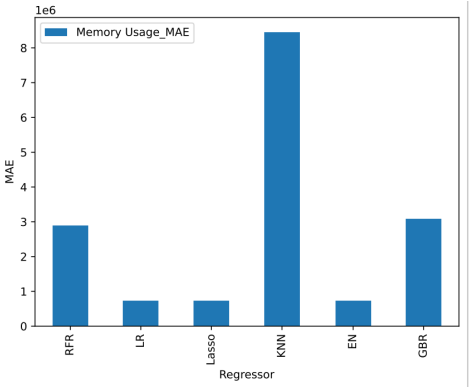

Fig. 20. Kmeans Regression - Memory Usage Performance - MAE

In terms of the R2 score and MAE, all models exhibit comparable total time performance.

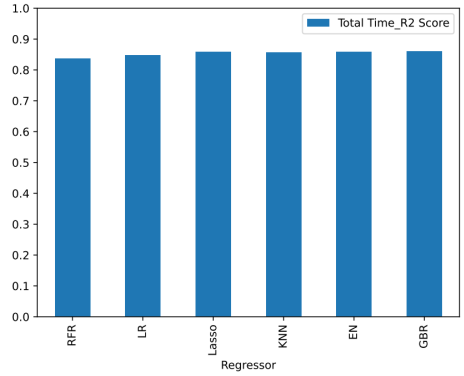

Fig. 21. Kmeans Regression - Total Time Performance - R2 Score

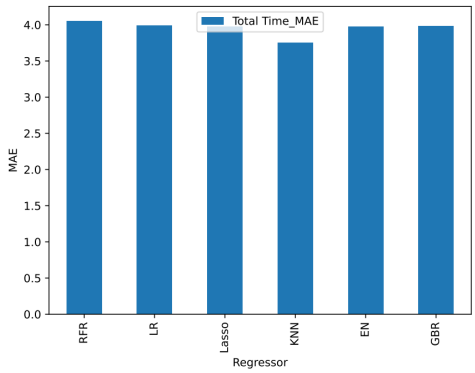

Fig. 22. Kmeans Regression - Total Time Performance - MAE

6) Random Forest Regression: Our R2 score analysis reveals that both Random Forest Regressor and Gradient Boosting Regressor perform exceptionally well, each explaining about 85% of the variability in the data. Linear Regression, Lasso Regression, and Elastic Net Regression also provide reliable results, explaining around 76% to 78% of the variance. KNN falls in between, with an R2 score of 0.80, indicating good predictive power.

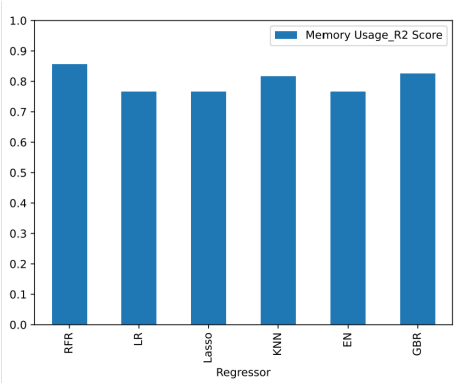

Fig. 23. Random Forest Regression - Memory Usage Performance - R2 Score

Our analysis based on MAE values reveals that the Random Forest Regressor and Gradient Boosting Regressor achieve the lowest average absolute prediction errors, each with an MAE of 5. Linear Regression, Lasso Regression, Elastic Net Regression, and KNN exhibit slightly higher MAE values of around 6.7 and 5.3, indicating a slightly larger average prediction error.

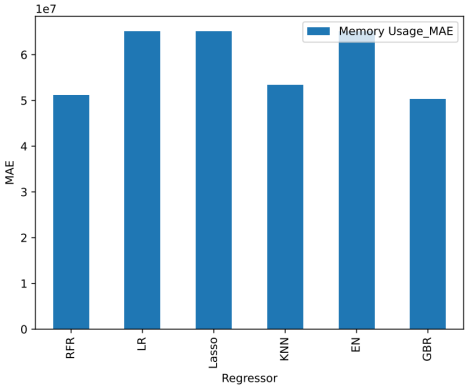

Fig. 24. Random Forest Regression - Memory Usage Performance - MAE

As far as total time prediction goes, the Random Forest Regressor excels with a remarkable R2 score of approximately 95%, while Linear Regression, Lasso Regression, and Elastic Net Regression also exhibit strong performance at around 84% to 85%. KNN achieves solid predictive accuracy with an R2 score of 0.87, and the Gradient Boosting Regressor performs robustly with an R2 score of 0.90.

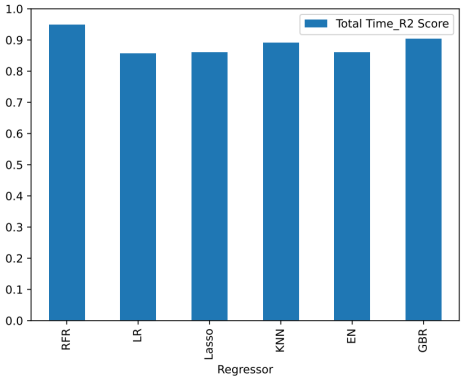

Fig. 25. Random Forest Regression - Total Time Performance - R2 Score

The Random Forest Regressor and Gradient Boosting Regressor stand out with impressively low MAE values of approximately 2.3, indicating strong predictive accuracy. Linear Regression, Lasso Regression and Elastic Net Regression exhibit slightly higher average prediction errors. KNN falls in between, performing well with a MAE of approximately 3.

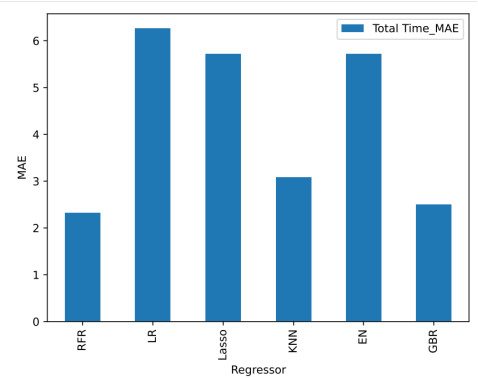

Fig. 26. Random Forest Regression - Total Time Performance - MAE

7) Word2Vec Regression: Our analysis of R2 score values indicates that the Random Forest Regressor (RFR), k-Nearest Neighbors (KNN), and Gradient Boosting Regressor (GBR) all yield R2 scores of 0, suggesting that these models struggle to explain the variance in the dependent variable. On the other hand, Linear Regression (LR), Lasso Regression (LASSO), and Elastic Net Regression (EN) show slightly improved, but still relatively low, performance with R2 scores of 0.1.

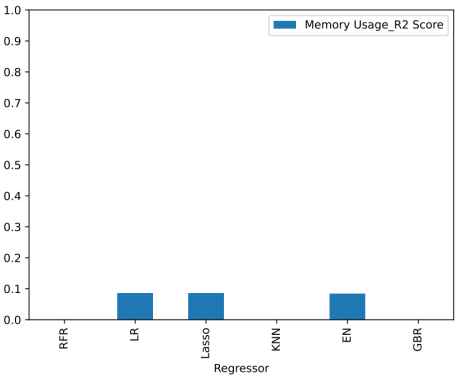

Fig. 27. Word2Vec Regression - Memory Usage Performance - R2 Score

As far as MAE is concerned, based on the histogram all models seem to perform admirably well, indicating strong predictive accuracy with minimal absolute prediction errors.

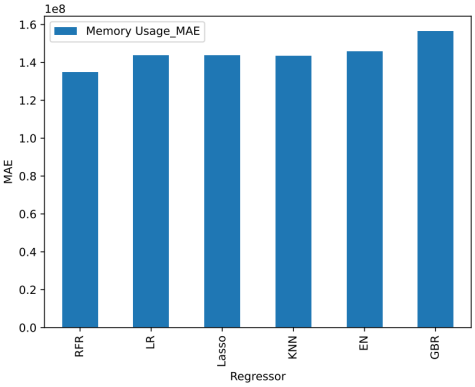

Fig. 28. Word2Vec Regression - Memory Usage Performance - MAE

Based both on R2 scores and MAE values we can conclude that while some models struggle to explain the variance in the dependent variable, they still manage to provide accurate predictions with minimal absolute errors. On the other hand, models with higher R2 scores tend to exhibit superior predictive accuracy and stronger explanatory capabilities.

Our R2 score analysis demonstrates outstanding performance in most models. The Gradient Boosting Regressor, while not perfect, still excels with a high R2 score, highlighting its strong predictive accuracy and explanatory power in capturing data patterns.

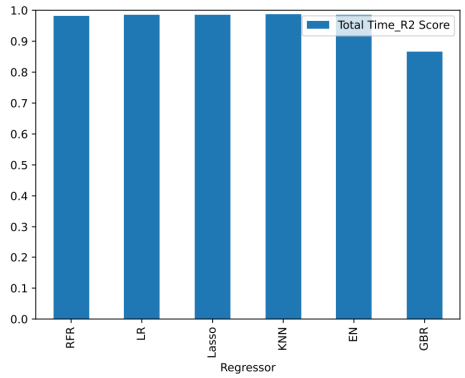

Fig. 29. Word2Vec Regression - Total Time Performance - R2 Score

In our examination of Mean Absolute Error (MAE) values, we observe notable variations in predictive accuracy across different regression models. The Random Forest Regressor as the top performer with the lowest prediction errors. The other models, while maintaining reasonable predictive accuracy, exhibit slightly higher MAE values. The Gradient Boosting Regressor, however, faces substantial challenges in minimizing prediction errors, as reflected in its higher MAE value.

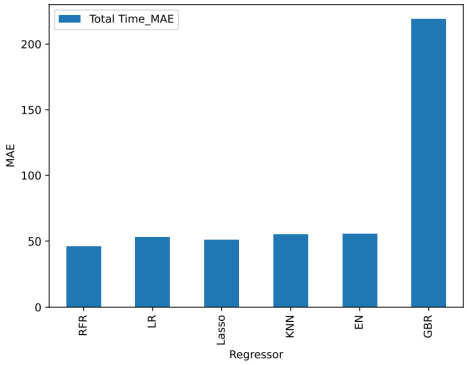

Fig. 30. Word2Vec Regression - Total Time Performance - MAE

IV. Conclusion

In this research endeavor, we embarked on a journey to explore the performance evaluation and modeling of machine learning operators within big data analytics runtimes, with a focus on Apache Spark. Our project encompassed various phases, including the installation and setup of the Spark cluster, the selection of operators for modeling, custom dataset generation, and rigorous performance measurement. The objective was to create prediction models that could anticipate operator behavior, such as execution times and resource utilization, without necessitating actual execution.

A. Key Findings

Throughout our experimentation and analysis, several key findings emerged:

1) **Custom Dataset Generation**: Designing and implementing our own data generators provided us with a high degree of control and flexibility. This approach allowed us to tailor datasets to diverse scenarios, encompassing varying dimensionalities and dataset sizes.
2) **Operator Diversity**: We selected a diverse set of machine learning operators, including k-means, Random Forest regression, and Word2Vec, for performance modeling. This diversity enabled us to assess the behavior of operators from different families under varying conditions.
3) **Modeling Accuracy**: Leveraging machine learning techniques, we created prediction models that accurately estimated operator performance metrics. These models exhibited minimal errors when applied to unseen data inputs, showcasing their effectiveness in predicting operator behavior.
4) **Efficiency and Resource Utilization**: Our careful configuration of the Spark cluster, including memory allocation, CPU cores, and garbage collection intervals, contributed to efficient resource utilization. This optimization was pivotal in ensuring smooth and reliable experiments.
5) **Dynamic Dataset Creation**: The ability to dynamically generate datasets with varying characteristics proved invaluable. It allowed us to explore a wide range of testing scenarios and ensured that our models could adapt to diverse input conditions.

B. Implications and Future Directions

Our research has several implications for the field of big data analytics and machine learning:

1) **Optimized Resource Allocation**: Our project highlights the significance of tailored resource allocation within big data analytics runtimes. Optimizing memory and CPU core allocation can substantially impact the efficiency of machine learning tasks.
2) **Model-Driven Predictions**: The successful development of prediction models for operator behavior opens the door to advanced resource management and task scheduling within distributed computing environments. These models can be utilized to make real-time decisions about task assignment and resource provisioning.
3) **Scalability Considerations**: Future research could explore the scalability of our approach to larger clusters

and more complex machine learning operators. Extending the project's findings to even more extensive datasets and diverse operator families would be valuable.

4) **Real-World Applications**: The prediction models developed in this study have the potential for practical deployment in real-world big data analytics scenarios. These models can assist in optimizing resource allocation, reducing execution times, and enhancing the overall efficiency of data processing tasks.

In conclusion, our project demonstrates the feasibility and effectiveness of modeling machine learning operator performance within Apache Spark. Through customized data generation, meticulous configuration, and advanced machine learning techniques, we have paved the way for enhanced resource management and predictive capabilities in big data analytics environments. Our findings contribute to the broader field of distributed computing and offer valuable insights into the optimization of machine learning workflows in large-scale data processing systems.

REFERENCES

[1] Apache Spark, *Apache Spark™ - Unified Engine for Large-scale Data Analytics*, https://spark.apache.org/

[2] Machine Learning Library (MLlib) Guide, https://spark.apache.org/docs/latest/ml-guide.html

[3] Set up Apache Spark on a MultiNode Cluster, https://medium.com/ymedialabs-innovation/apache-spark-on-a-multi-node-cluster-b75967c8cb2b

[4] PySpark 3.4.1 Official Documentation, https://spark.apache.org/docs/latest/api/python/index.html

[5] Pyhton 3.8 Installation on Ubuntu Guide, https://spark.apache.org/docs/latest/api/python/index.html

[6] Natural Language Toolkit Official Documentation, https://www.nltk.org/