A Case Study: Applied Regression Analysis Using Distributed Machine Learning

Eric Wang*
e246wang@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

Jacky Chen*
j57chen@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

ABSTRACT

In recent years, more and more enterprises and individuals are applying machine learning methods to help extract intensive insights into business and investment strategies, thanks to the rapid development of Internet technologies that have led to the unprecedented growth in data volumes. For example, the used car market has become increasingly popular as buyers seek more affordable options and sellers look to offload their vehicles. The mining of these valuable data can help consumers and traders better determine reasonable transaction prices. However, the establishment of the regression models on the huge amount of transaction data in the used car market is beyond the reach of traditional single node computing. Besides, determining the fair market value of a used car can also be challenging, as it depends on various factors such as the car's make and model, year of manufacture, mileage, and overall condition. In this project, we propose the application of different machine learning methods over distributed computing platforms and hire a massive used car transaction dataset as an instance to perform the whole process of big data analysis, including data pre-processing, exploratory data analysis, feature engineering and machine learning modeling using Spark and Spark ML. Specifically, in the proposed system, missing values are imputed in one dataset and the results are compared with another non-imputed dataset. In categorical data conversion, one-hot encoding is used to represent the feature availability. Finally, we evaluated our method quantitatively and qualitatively and discussed the pros and cons of each technique involved. By practicing the manipulation of the massive dataset on distributed computing platform, this project not only serves the future learners as a comprehensive guide to distributed machine learning and big data analysis, but also provides buyers and sellers with a tool to better estimate the value of a used car and delivery data-driven decisions.

KEYWORDS

regression models, distributed computing, machine learning, data analysis

1 INTRODUCTION

Machine learning methods are increasingly being used to analyze massive dataset and build decision-making systems. Due to the complexity of problems, such as self-driving cars control, speech recognition or consumer behavior prediction, traditional single node computing and training solutions are not feasible enough. A typical scenario is that centralized solutions do not work when a large enterprise's data is inherently distributed or the volume of data is too large to be stored on a single node. Furthermore, in a production environment, the long running time of model training on a single node also motivates solution designers to use distributed systems to increase the degree of parallelism and the total amount of IO bandwidth. In order for these distributed structured data to be accessible as training data for machine learning problems, algorithms must be selected and implemented to be able to compute in parallel, accommodate multiple data distributions, and be resilient to failures.

On the other hand, the boundaries between traditional supercomputers, grid and cloud are increasingly blurred, especially for the optimal execution environment demanding heavy workloads such as machine learning tasks. For example, GPUs and accelerators are commonly found in major cloud data centers. Therefore, parallelization of machine learning workloads brought by hardware is critical to achieve acceptable performance at scale. However, distributed machine learning faces serious algorithmic challenges in terms of performance, scalability, failure resilience, or security when transitioning from centralized solutions to distributed systems. As with other large-scale computing challenges, there are mainly two fundamentally different and complementary ways to accelerate the distributed machine learning workloads. One is to add more resources to a machine (vertical scaling, such as increasing GPU/TPU computing cores), which can be more classified into the domain of High Performance Computing (HPC). The second one is to add more nodes to the computing system (horizontal scaling and low cost), which is the topic we mainly focus on in this project.

Distributed machine learning, also known as distributed learning, refers to algorithms and systems that use multiple computing nodes for machine learning or deep learning. It aims to improve performance, protect privacy, and can be extended to larger training data and larger model. The storage and optimization methods for distributed machine learning will be introduced more in the following sections. Particularly, in this project we discuss more about traditional machine learning methods, such as Generalized Linear Models (GLMs), Gradient Boosted Trees (GBTs) and XGBoost. On one hand, we believe that model interpretability is very important in big data analysis. For example, for the regression model built on the dataset we used, knowing which features contribute more to the change in used car prices is very helpful for subsequent analysis and business decisions. On the other hand, when training a complex neural network on a distributed computing platform, not only the training time is way longer than that of traditional methods, but also the problem of memory overflow often exists when there are too many model parameters. For example, deploying large-scale machine learning tasks using Spark often requires the use of Resilient Distributed Datasets (RDDs) to accommodate all model

*Both authors contributed equally to this project.

parameters. In order to accommodate updated model parameters, new RDDs generally need to be created in each iteration, which is a performance bottleneck for iterative operations in machine learning.

Recently the used car market has experienced significant growth and popularity. This trend can be attributed to several factors, including the increasing cost of new cars, the availability of financing options for used cars and the general economic climate. Consumers are also becoming more interested in sustainable and environmentally friendly options, leading to a rise in demand for used electric and hybrid vehicles. Additionally, the COVID-19 pandemic has caused disruptions in the automotive industry, resulting in supply chain challenges and shortages of new cars, further driving up the demand for used cars. As a result, the used car market has become more competitive, with dealerships and private sellers alike seeking to capitalize on the trend.

Determining the fair market value of a used car can be challenging, as it depends on various factors such as the car's make and model, year of manufacture, mileage, and overall condition. In this project, we propose the development of a regression model to predict the price of used cars, based on a variety of relevant factors. By building a robust and accurate model, we hope to provide buyers and sellers with a tool to better estimate the value of a used car and delivery data-driven intelligence. We will use a dataset of used car sales [15], including information on car features, mileage, and price, to train and test our regression model. We will explore different types of regression models, such as linear regression, decision trees, and random forests, and evaluate their performance in predicting the prices of used cars. We will also explore different techniques for feature selection and model optimization, in order to build the most accurate, efficient, and scalable model as much as possible. Overall, this project aims to contribute to the growing field of predictive analytics and to provide insights into the factors that affect used car prices. Our project is publicly available on GitHub at https://github.com/jacky1c/A-Case-Study-Applied-Regression-Analysis-Using-Distributed-Machine-Learning

2 RELATED WORKS

Several related works have been done on the subject of used car price prediction. Even though some of the works achieved impressively low error, the datasets used in their studies are too small to make their studies universally significant. For example, Noor et al. [16] used only 1699 records and achieved the best R-square of 0.98 with linear regression model. Pandit el al. [17], as another example, experimented with generalized linear models, decision tree and XGBoost[2] on 4340 rows of data, and achieved the highest R-square of 0.95.

Jin [10] obtained 13,120 Mercedes cars data and trained several regression models, including linear regression, polynomial regression, support vector regression, decision tree regression, and random forest regression. Among all five regression models, random forest regression achieved the highest R-square of 0.90.

Cui et al. [3] proposed the DXL algorithm where ResNet [9] was combined with an iterative framework of XGBoost [2] and LightGBM [11]. The first step of DXL algorithm was to combine ResNet output with the original feature to form a new set of features. Then the combined features were used by XGBoost and LightGBM to generate new features until the best result was generated. Their model was trained on 30,000 training data, achieving R-square of 0.75.

Some preliminary data exploratory analysis have been done on the same dataset we choose [15]. Valev [22] analyzed the distributions of features in detail. Kim [12] adopted Vaex [21] as the big data framework to process the dataset. Vaex helps with processing large datasets efficiently and swiftly by lazy computations, virtual columns, memory-mapping, zero memory copy policy, and efficient data cleansing. It is shown to be faster than PySpark at processing big data in the performance benchmark report conducted by Alexander [1].

Lozano et al. [13] achieved the lowest MAE of 4189.33 using CatBoost regression [18] on the same dataset. In addition, they found horsepower, mileage, torque and year were the most important features to the model.

Based on our literature review, we notice that as the dataset size increases it is harder for models to achieve high accuracy, because larger datasets provide more noises of the problem domain. On the other hand, larger datasets provide more diverse and universally representative examples, which allow models to be more robust. To our best knowledge, Spark has not been used as the big data framework to analyze this particular dataset.

3 DISTRIBUTED MACHINE LEARNING

In a production environment, the efficiency of product development, maintenance and updates is often subject to the amount of data involved in the product. In particular, large-scale data processing, modeling, and analysis often demand a big data platform to solve the problem of distributed training of machine learning models, which is quite prominent in the fields of recommendation systems, advertisement targeting, and information retrieval. A series of Python libraries with interactive programmings, including scikit-learn, numpy, SciPy, pandas and Matplotlib, are more suitable for rapid development of prototypes and feasibility analysis. However, under the context of the Internet, the amount of terabytes or even petabytes of data makes it almost impossible to use a single node to complete the training of machine learning models. Therefore, it is a better choice to use distributed machine learning training. Specifically, supporting Python, Java, Scala and R at the same time, the scalability and adaptability of Spark platform make future product development iterations easier, which is the most primary reason why we use Spark in every stage of this project, including data pre-processing, feature engineering, modeling, analysis and evaluation.

3.1 Distributed Data Storage for Learning

Parameter updating and learning of machine learning models tends to be more run on a single node with a small amount of data. Meanwhile, a basic assumption of machine learning is the independence of samples. But in practical applications, it is often difficult for a limited amount of data to guarantee a high degree of independence among samples, resulting in inaccurate learned model with unsatisfactory performance. One of the advantages of distributed file systems such as Hadoop Distributed File System (HDFS) in the field of machine learning is that it is possible to both store massive amounts of data and perform machine learning on the full amount of data, which is critical to solving the problem of insufficient statistical independence and randomness.

On the other hand, due to the natural limitations of MapReduce, the use of MapReduce to implement distributed machine learning algorithms is very time-consuming and highly costs disk IO. Usually, the process of machine learning algorithm parameter learning requires a large number of iterative calculations, that is, the result of a certain calculation would be used as the input for the next iteration. In this process, the operation logic of MapReduce requires that the intermediate results be stored on the disk and re-read during the next calculation, which is an obvious performance bottleneck for algorithms highly dependent on iterative calculations. However, the memory calculation characteristics of Spark are naturally suitable for iterative computing. In particular, Spark ML provides a machine learning library, for massive data manipulation, that includes the distributed implementation of commonly used machine learning algorithms to strike a balance between iteration calculation efficiency and data storage scale as much as possible. Spark ML Pipeline API can also easily combine data processing, feature transformation, regularization and different machine learning algorithms to build a single complete machine learning pipeline.

3.2 Parallel Computing for Optimization

Having introduced the advantages of Spark distributed computing in machine learning, we further discuss the principle of Spark ML parallel training.

Note that not every machine learning model is suitable for distributed computing. For example, the model structure of Random Forest allows it to completely perform model training of parallel data, while the structure of Gradient-boosted Decision Trees (GBDT) requires that only serial training be performed among trees. However, in the fields of data science and artificial intelligence, machine learning methods that use gradient descent and its variants as optimization methods have always remained the mainstream. Therefore, in different machine learning methods, we will focus more on the distributed computing implementation of gradient descent-dependent methods, because the implementation quality of the degree of parallelism for gradient descent dominates the training efficiency of these machine learning and deep learning methods, such as Logistic Regression and Multiple Layer Perceptron.

Here, we dig into the source code of Spark ML and investigate the mini-batch gradient descent implementation as shown in Algorithm 1, where only the most critical snippet of pseudo-code are listed.

Algorithm 1 snippet of runMiniBatchSGD

```
1: while The number of iterations has not exceeded the limit do
2:     Broadcast all weight parameters of the model.
3:     Calculate the gradients after sampling in each node, and
       summarize the gradients through treeAggregate().
4:     Update weights based on the gradients.
5:     Increment the number of iterations.
6: end while
```

Specifically, the code line 2 actually broadcasts the current model parameters to each data partition, which can be used as a virtual computing node. The code line 3 means that each computing node performs data sampling to obtain the data of the mini batch and calculates the gradients separately. Then the gradients are summarized using the aggregation function and the final gradients are obtained. After the number of iterations has reached the limit or the model has sufficiently converged, the model stops training. The above operations can be summarized as the whole process of Spark ML performing mini batch gradient descent, which is also a typical example of implementing distributed machine learning optimization.

To summarize, the distributed training process of distributed machine learning is actually a “data parallel” process, which does not involve an overly complicated gradient update strategy, nor does it implement parallel training through parameter parallelism. This method is simple, intuitive and easy to implement.

In the following sections, we use Spark and Spark ML performing the whole process of big data analysis, including exploratory data analysis, pre-processing, modeling and evaluation, on the used car dataset. We adopt a standard process for machine learning projects described in *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* [8].

4 ANALYSIS

4.1 Data Acquisition

U.S. used cars dataset is obtained from Kaggle [15] in CSV file format. This dataset contains over 3 million used car records and 66 attributes. It was obtained by running a crawler on Cargurus inventory in September 2020. In addition, we downloaded a lookup table from Simple Maps [14] to map U.S. cities to states, and a U.S. states geographical boundary dataset from Solanki’s GitHub [20]. The additional data is used in this project to group cities by corresponding states and to create data visualizations.

4.2 Exploratory Data Analysis

In this study, price is the response variable and the other attributes are examined in exploratory data analysis process to check the significance for predicting the price. The lowest price in dataset is less than a thousand dollars and the prices for luxury cars go over a million dollars, with the most expensive car being Ferrari Enzo, worth of over three million dollars.

Exploratory data analysis is developed on a copy of small sample dataset and then scaled up to the whole population dataset. To trace the records in the sample dataset back to the population dataset, we add row number as surrogate key before down sampling. The natural primary key (listing ID) is not selected as unique key because listing ID could contain duplicated values in different source data systems and the source systems could change in the future.

4.2.1 Missing values. We calculate the percentage of missing values for every column. 16 columns have over 45% of missing values, whereas 18 columns have no missing values. 26 columns have less than 7% of missing values. The proportion of missing values highly affects the usefulness of the features.

4.2.2 Outliers. Outliers are spotted in this dataset. For example, a 2018 Ford Edge was sold for 2.6 million dollars in Nova Scotia, and a 2020 Chevrolet Impala was sold for 2.2 million dollars in Kansas. Given that most of the data points seem normal, we are confident that there are enough normal data points so we decide to proceed our analysis without removing outliers.

4.2.3 Numeric features. We performed a comprehensive data transformation to parse numeric features from strings as many as possible before analyzing numeric features. A correlation matrix is generated to analyze the relationships between price and other numeric features, as shown in Figure 1. We notice that features related to vehicle power (horsepower, torque and number of engine cylinders), size (height, width, length and wheelbase) and vehicle usage (year and mileage) are highly correlated with price. In contrast, fuel economy, savings amount and seller rating have relatively low impact on the price. An interesting observation is that consumers pay more attention to back legroom than front legroom.

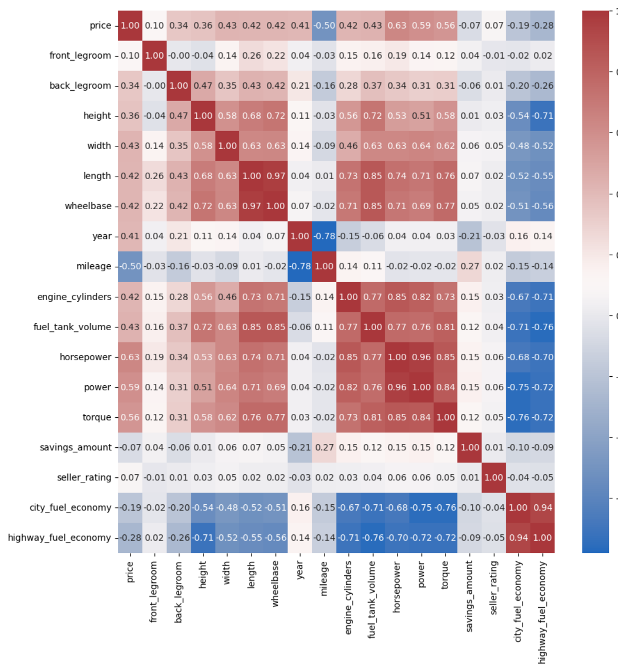

Figure 1: Correlation matrix heat map

4.2.4 Categorical features. There are more categorical features than numeric features in our dataset. We analyzed the distributions for some of the features that we think are significant to predicting price. The top 3 most popular body types are SUV, which is around half of the used cars, followed by sedan and pickup truck. Black and white are the most popular exterior colours in our dataset, being 20% respectively, followed by silver and gray. In addition, we visualized the average price by states as shown in Figure 2. We noticed that the average price in Hawaii is over 50% higher than the second most expensive state. To make the graph more readable, Hawaii is not included in this graph.

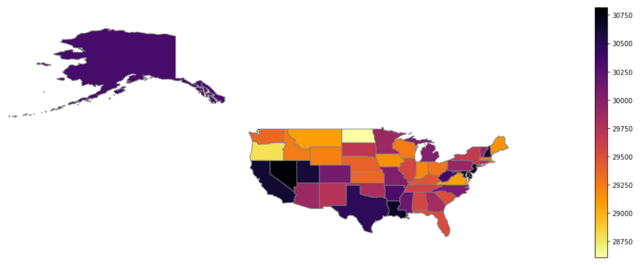

Figure 2: Mean price by state (without Hawaii)

4.3 Data Preparation

4.3.1 Data cleaning. To reduce wastefulness of data, missing values are imputed informatively using the analysis performed above. For example, missing mileage is filled with average mileage based on the state. If the state is unknown, we fill mileage with a nation-wise average.

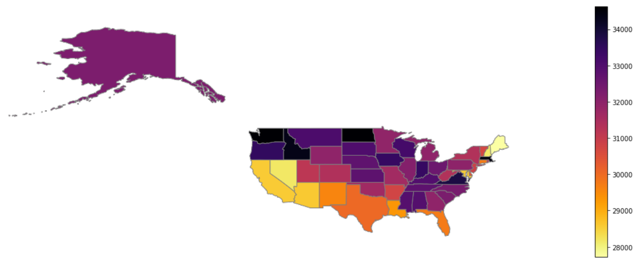

Figure 3: Mean mileage by state (without Hawaii)

4.3.2 Feature selection. 6 columns that are related to ID, description and URL are dropped because they are of little use for predicting price or filling missing values. Moreover, 16 columns with over 45% missing values are dropped. Among these columns, bed size, bed height, bed length and cabin size are only applied to pickup trucks and do not pass correlation test. Some of the categorical columns would haven been useful if there were more data, such as vehicle damage category, whether the vehicle is certified, and whether the vehicle was stolen and later recovered. Maximum seating, highway fuel economy and city fuel economy are not selected because they are not significant for price prediction as shown in the correlation matrix.

4.3.3 Feature engineering. Categorical features are firstly converted into numerical format using string indexer [6], which orders categorical data by frequency in descending order and then assigns continuous index to strings starting from 0. Then, indices are converted into one-hot binary vectors using one hot encoder [4]. In terms of high cardinality features, we perform extra steps to reduce one-hot encoder dimensions before feeding them to the models. For example, cities are grouped into states, reducing thousands of unique values to 51 unique values (including N/A for cities that cannot be mapped to a state). Car model names, as another example, are highly dependent on car makes, body type, power, torque, etc. So we replaced car model names by a combination of other features, reducing vector dimension from 1426 to 113. Based on our analysis, top 10 most popular car makes occupy 70% of the dataset, as shown in Figure 4. We could group low popular makes as other to further reduce dimension to 23.

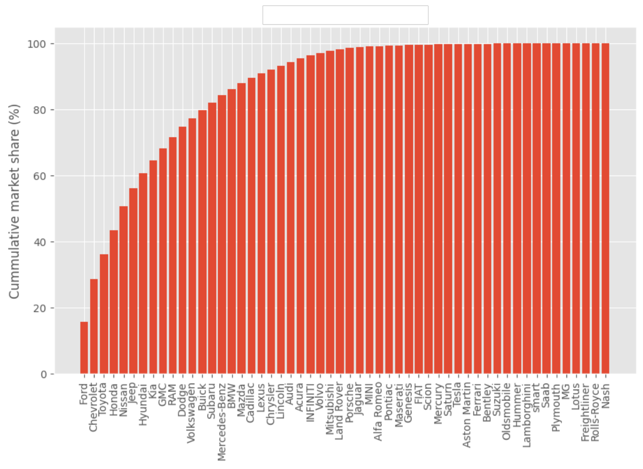

Figure 4: Cumulative market share by makes

4.3.4 Feature scaling. We adopted the standard scaler transformation [5], which normalizes each feature to have unit standard deviation and zero mean. We observed a slight improvement of model performance for generalized linear models, and an insignificant decrease for tree based models.

4.4 Regression Models

4.4.1 Generalized Linear Models. Generalized Linear Models (GLMs) extend the linear model by allowing the specification of a link function and a probability distribution that relates the mean of the response variable to a linear combination of the independent variables. The most common GLM is linear regression which uses Gaussian distribution and identity link function. Since the response variable in the dataset is non-negative, Gaussian and Poisson distribution are two suitable probability distributions for GLMs.

4.4.2 Gradient Boosted Trees. Gradient Boosted Trees (GBTs), based on the stochastic gradient boosting algorithm developed by Friedman [7], are a type of ensemble learning algorithm that combines multiple decision trees to improve the predictive power of a model. In GBTs, decision trees are trained sequentially, with each new tree trying to correct the errors made by the previous trees. This is done by fitting the new tree to the residual errors of the previous tree. The trees are added one at a time, and the process continues until a specified stopping criterion is met, such as a maximum number of trees or a minimum improvement in the model's performance.

4.4.3 XGBoost. eXtreme Gradient Boosting (XGBoost), developed by Chen et al. [2], is another machine learning model that is based on the gradient boosting algorithm, designed to be more efficient and scalable than GBTs. Specifically, XGBoost used a more regularized objective function to control over-fitting, which gives it a better performance. Two additional techniques are used to further prevent over fitting. The first technique is shrinkage, which scales newly added weights by a factor after each step of tree boosting. The second technique is feature sub-sampling, which not only prevents over-fitting, but also speeds up computations of the parallel algorithm.

Another important improvement in XGBoost is the sparsity aware algorithm. Sparsity aware algorithm exploits the sparsity to make computation complexity linear to number of non-missing entries in the input. Our dataset is sparse due to the artifacts of one-hot encoding of categorical features, so sparsity aware algorithm is suitable to reduce running time.

One of the efficiency issues in tree learning algorithms is the exact greedy algorithm. It enumerates over all the possible splits on all the features to find the best split of the decision tree. However, it is impossible to efficiently do so when the data does not fit entirely into memory. To enable distributed computing and speed up algorithm's training process, XGBoost uses a technique called histogram-based approximation, where the algorithm bins the feature values into discrete intervals or buckets. This allows the algorithm to calculate the gradient statistics and find the best split points more efficiently, without having to look at every individual feature value.

4.5 Model Evaluations

For each model mentioned in the previous section, we experimented with different choices of feature selections. Furthermore, we fine-tuned model parameters using grid search and 5-fold cross validation. The model performances shown in Table 1 are measured using the best feature selection on test dataset. Among 8 regression models that we evaluated, XGBoost achieves the best R-square score (0.89) and the lowest MAE (3093.84). 11 features are used in the best XGBoost model, including body type, make name, transmission display, listing color, state, franchise make, year, mileage, horsepower, engine cylinders and fuel tank volume. However, we ran into memory issues when training XGBoost model with about 200,000 rows of data in Google Colaboratory basic environment.

Compared to XGBoost, GBT model is less accurate and requires longer time to train. Even though GLM with Gaussian distribution and identity link function, commonly known as linear regression, achieves higher error than XGBoost, it is significantly faster and requires less memory than tree based models. In addition, the features of the best linear regression are 3 less than the features used in XGBoost. Therefore, linear regression could be considered as an alternative model when resource is limited.

To achieve the best prediction result, XGBoost is chosen as the model to be deployed in this project, and model deployment will be introduced in the next section.

5 MODEL DEPLOYMENT

XGBoost failed to train on large dataset in Google Colaboratory environment because of the Google Colaboratory's single node limitation. To fully leverage the distributed nature of XGBoost, we established a Spark cluster environment on Amazon Web Services (AWS). In this section, we will discuss the advantages of AWS Elastic MapReduce (EMR) and how it is setup for our project.

Table 1: Experiment Results. The possibility distribution and link function of GLMs are specified in the parentheses. Training time was measured on 20,000 rows of training data in Google Colaboratory basic environment (Intel® Xeon® 2-core CPU, 12 GB RAM).

Model	R-square	RMSE	MAE	Training time
GLM (Gaussian + Identity)	0.75	8407.35	5124.23	1s
GLM (Gaussian + Log)	0.64	10160.27	6200.69	1s
GLM (Gaussian + Inverse)	-3984609307.51	1063082810.26	1063081022.18	1s
GLM (Poisson + Identity)	-1.1	31428.95	26781.25	1s
GLM (Poisson + Log)	0.62	10383.77	6115.22	1s
GLM (Poisson + Sqrt)	0.66	9805.29	5823.29	1s
GBT	0.85	6481.99	3476.18	34s
XGBoost	**0.89**	**5775.91**	**3093.84**	11s

5.1 AWS EMR

EMR is an AWS-managed cluster platform that supports big data frameworks, including Apache Hadoop and Spark. EMR also supports transforming and loading large amounts of data into and out of other data stores and databases within AWS ecosystem, such as Amazon Simple Storage Service (Amazon S3) and Amazon DynamoDB. For this project, AWS S3 is chosen as the data storage due to its simplicity of use. EMR cluster supports 3 types of nodes, namely primary nodes, core nodes and task nodes. The differences among them will be covered in the sub-sections below.

5.1.1 Primary nodes. The primary node manages the overall EMR cluster and coordinates the distributed processing of data. It runs the YARN Resource Manager service to manage the allocation of resources, such as CPU and memory, to each task based on the requirements of the job. It also runs the Hadoop Distributed File System (HDFS) Name Node service, tracks the status of jobs submitted to the cluster, and monitors the health of the instance groups.

5.1.2 Core nodes. Core nodes are managed by the primary node and they are required for every EMR cluster. They are responsible for processing and storing data, and are typically used to run Hadoop Distributed File System (HDFS) and other distributed processing frameworks, such as Apache Spark. With instance groups, Amazon EC2 instances can be added or removed while the cluster is running. One can also set up automatic scaling to add instances based on the value of a metric.

5.1.3 Task nodes. Task nodes, on the other hand, are used for processing tasks that are assigned by core nodes and they are optional. However, tasks nodes are not responsible for storing data to HDFS so data on task nodes will be lost if they are terminated. One of the use cases of task nodes is to process streams from S3. In this case, network IO will not increase as the used data is not on HDFS. As with core nodes, you can add task nodes to a cluster by adding Amazon EC2 instances to an existing uniform instance group or by modifying target capacities for a task instance fleet.

5.2 Environment Configuration

To support running XGBoost on millions of rows of data, we created an EMR cluster with 1 primary node and 10 core nodes, one XGBoost worker per core node. More specifically, the primary node is an EC2 m5.xlarge instance and the core nodes are EC2 c5.large instances [19], the cheapest instances in C5 families, which have Intel® Xeon® 2-core CPU and 4 GB memory. The C5 instance family is designed for running advanced compute-intensive workloads, such as distributed analytics and machine learning inference.

Once the cluster is created, 2 steps are created to perform machine learning tasks. The first step, responsible for loading and cleaning data and feature engineering, will be triggered when there is a training dataset uploaded to S3 bucket, as shown in Figure 5. The output of the first step is a Parquet file containing features in a vector form. The second step is to load the Parquet file and train XGBoost model. The outputs of the second step are model artifacts stored in S3.

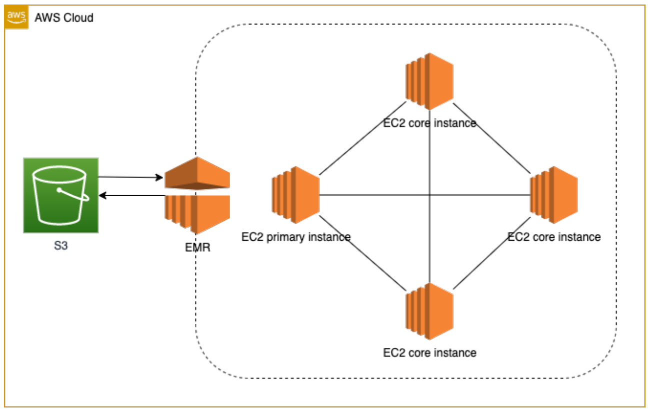

Figure 5: AWS environment configuration. EMR cluster has 1 primary node and 3 core nodes and is connected to an S3 bucket.

5.3 Advantages of Cloud-Based Distributed Computing

Cloud-based distributed computing allows for easy scaling up or down of computing resources as needed, which can help enterprises save costs and improve performance. In addition, little maintenance is needed for hardware and software updates. Furthermore, it provides high availability of services, as data is distributed across multiple servers and data centers, reducing the risk of downtime or service interruptions.

Overall, distributed computing in the cloud is beneficial for businesses to efficiently manage large-scale machine learning projects due to its high scalability and availability, as well as low maintenance efforts.

6 LIMITATIONS

Distributed data storage and computing indeed help solve the problem of machine learning in terms of de-centralized massive data distribution and contribute to parallelism for optimization, but there still have been some limitations we found when we performed the data analysis for this project using Spark.

Firstly, Spark adopts a blocking gradient descent method, and each round of gradient descent is determined by the slowest node. In other words, if a node takes too long to calculate the gradient due to problems such as data skew, this process will block all other nodes from performing new tasks. This distributed gradient computation approach leads to low parallel training efficiency.

If we develop Spark to train neural network models, the global broadcast approach would broadcast all model parameters before each iteration, which is a very bandwidth-intensive method, especially with the model possessing a large number of parameters. The broadcast process and the process of maintaining a copy of the weight parameters at each node are highly resource-consuming, which leads to Spark's undesired performance in the face of complex models.

Spark only supports the training of standard multi-layer perceptron neural networks, so it is difficult to achieve complex network structures with a large number of adjustable hyperparameters and activation functions. This is the biggest reason that restricts us from using distributed computing to train large-scale deep learning models. In future practice, we will try to improve Spark and apply parameter parallelism, instead of just data parallelism, to distributed machine learning. We believe that in the era of deep learning with large models and massive parameters, this is an excellent outset to efficiently work on more complex data science and artificial intelligence tasks.

7 CONCLUSION

In this project, we performed comprehensive data analysis, including data pre-processing, exploratory data analysis and feature engineering, on large used cars dataset and experimented with various regression models to predict the price. XGBoost model achieves the best MAE of 3093.84, a 26% of improvement than the work done by Lozano et al. [13] on the same dataset. Furthermore, we deployed the XGBoost model in a cloud-based distributed environment using AWS EMR and S3, which enables XGBoost algorithm to process millions of rows of data in parallel.

REFERENCES

[1] Jonathan Alexander. 2023. Beyond pandas: spark, dask, vaex and other big data technologies battling head to head. (Mar. 2023). Retrieved March 8, 2023 from https://towardsdatascience.com/beyond-pandas-spark-dask-vaex-and-other-big-data-technologies-battling-head-to-head-a453a1f8cc13.

[2] Tianqi Chen and Carlos Guestrin. 2016. XGBoost. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, (Aug. 2016). DOI: 10.1145/2939672.2939785.

[3] Baoyang Cui, Zhonglin Ye, Haixing Zhao, Zhuome Renqing, Lei Meng, and Yanlin Yang. 2022. Used car price prediction based on the iterative framework of xgboost+lightgbm. *Electronics*, 11, 18. https://www.mdpi.com/2079-9292/11/18/2932.

[4] The Apache Software Foundation. 2023. Pyspark one hot encoder. (Mar. 2023). Retrieved March 28, 2023 from https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.OneHotEncoder.html.

[5] The Apache Software Foundation. 2023. Pyspark standard scaler. (Mar. 2023). Retrieved March 28, 2023 from https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.StandardScaler.html.

[6] The Apache Software Foundation. 2023. Pyspark string indexer. (Mar. 2023). Retrieved March 28, 2023 from https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.StringIndexer.html.

[7] Jerome H. Friedman. 2002. Stochastic gradient boosting. *Comput. Stat. Data Anal.*, 38, 367–378. DOI: 10.1016/S0167-9473(01)00065-2.

[8] Aurélien Géron. 2019. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. (2nd ed.). O'Reilly Media, Inc., 1005 Gravenstein Hwy N, Sebastopol, CA, USA. ISBN: 9781492032649.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. (2015). arXiv: 1512.03385 [cs.CV].

[10] Chuyang Jin. 2021. Price prediction of used cars using machine learning. In *2021 IEEE International Conference on Emergency Science and Information Technology (ICESIT)*, 223–230. DOI: 10.1109/ICESIT53460.2021.9696839.

[11] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: a highly efficient gradient boosting decision tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (NIPS'17). Curran Associates Inc., Long Beach, California, USA, 3149–3157. ISBN: 9781510860964.

[12] Seungjun Kim. 2023. Gentle intro to vaex, the rising big data lib. (Mar. 2023). Retrieved March 8, 2023 from https://www.kaggle.com/code/juminator/gentle-intro-to-vaex-the-rising-big-data-lib.

[13] Alvaro Lozano, Andrea Lizondo, Mario Muriel, Pablo Abad, and Pablo Casero. 2023. Mileage - vehicle's price predictor. (Mar. 2023). Retrieved March 8, 2023 from https://www.kaggle.com/code/alvarolozanoalonso/mileage-tfm-15-09-2021.

[14] Simple Maps. 2023. United states cities database. (Mar. 2023). Retrieved March 28, 2023 from https://simplemaps.com/data/us-cities.

[15] Ananay Mital. 2023. Us used cars dataset. (Mar. 2023). Retrieved March 8, 2023 from https://www.kaggle.com/datasets/ananaymital/us-used-cars-dataset.

[16] Kanwal Noor and Sadaqat Jan. 2017. Vehicle price prediction system using machine learning techniques. *International Journal of Computer Applications*, 167, 9, (June 2017), 27–31. DOI: 10.5120/ijca2017914373.

[17] Eesha Pandit, Hitanshu Parekh, Pritam Pashte, and Aakash Natani. 2022. Prediction of used car prices using machine learning techniques. *International Research Journal of Engineering and Technology*, 09, 9, (Dec. 2022), 355–360. DOI: 10.5120/ijca2017914373.

[18] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. 2019. Catboost: unbiased boosting with categorical features. (2019). arXiv: 1706.09516 [cs.LG].

[19] Amazon Web Services. 2023. Amazon ec2 c5 instances. (Mar. 2023). Retrieved March 28, 2023 from https://aws.amazon.com/ec2/instance-types/c5/.

[20] Sunny Solanki. 2023. United states geographical data. (Mar. 2023). Retrieved March 28, 2023 from https://github.com/sunny2309/datasets/blob/master/us-states.json.

[21] vaex.io. 2023. What is vaex? (Mar. 2023). Retrieved March 8, 2023 from https://vaex.io/docs/index.html#.

[22] Valcho Valev. 2023. Data analysis - used$_cars_data_3million_usa$. (Mar. 2023). Retrieved March 8, 2023 from https://www.kaggle.com/code/valchovalev/data-analysis-used-cars-data-3million-usa.