



Available online at <http://scik.org>

J. Math. Comput. Sci. 11 (2021), No. 5, 5267-5277

<https://doi.org/10.28919/jmcs/5931>

ISSN: 1927-5307

IMPLEMENTING MACHINE LEARNING ALGORITHMS ON SPARK

SHWETA MITTAL*, OM PRAKASH SANGWAN

Department of Computer Science, Guru Jambheshwar University of Science & Technology, Hisar, Haryana, India

Abstract: Massive amount of data is being generated from the number of sources on day to day basis. Spark is a very popular open source platform available freely on web to store and process big databases. For training the machines to learn hidden patterns/information from these huge raw databases, machine learning algorithm needs to be implemented. ML and MLlib are two machine learning libraries to implement machine learning algorithms in Spark. In this paper, Decision Trees, Random Forests and Gradient Boosted Trees have been implemented by using Cardiac and Telecom dataset on local PC as well as Google Colab and it was concluded that Gradient Boosted Trees performed better than Decision Trees and Random Forests in terms of accuracy but took longer time to execute. Further, it has been also observed that algorithms took less time to run on Colab GPU as compared to local PC.

Keywords: machine learning; spark; MLlib; decision trees; big data.

2010 AMS Subject Classification: 68W40.

1. INTRODUCTION

Tremendous amount of data is being generated via number of domains i.e. Health Sector, Banking, Educational Institutions, Social Media etc. on day-to-day basis. Spark is open source platforms developed by Apache to store and process such big databases which can support data in form of text as well as images. The fundamental data structure of Spark is RDD i.e. Resilient

*Corresponding author

E-mail address: shwetamittal019@gmail.com

Received April 26, 2021

Distributed Dataset which is read-only dataset distributed over numerous nodes. In Spark, there is one master node to control the entire process of resource allocation, job scheduling, task management etc. and several worker nodes to perform the job. Spark performs in-memory computing, thus is faster than Hadoop and supports iterative algorithms. Various platforms to implement Spark cluster are as follows: Azure Databricks (built on the top of AWS, Amazon Web Services), Amazon Elastic Map Reduce, Google Cloud, Dockers etc.

Machine learning is a technique to train the system to learn from the data and to make predictions from it. There are 2 libraries available in Spark i.e. ML and MLLib to implement machine learning algorithms. MLLib is a primary API for Spark and is built on the top of RDDs (Resilient Distributed Datasets) whereas ML library is built on the top of Dataframes. Logistic Regression, Decision Trees, Random Forest, Gradient Trees, Multi-Layer Perceptron, Naïve Bayes, SVM are some of the machine learning techniques supported by Spark (as shown in Figure 1).

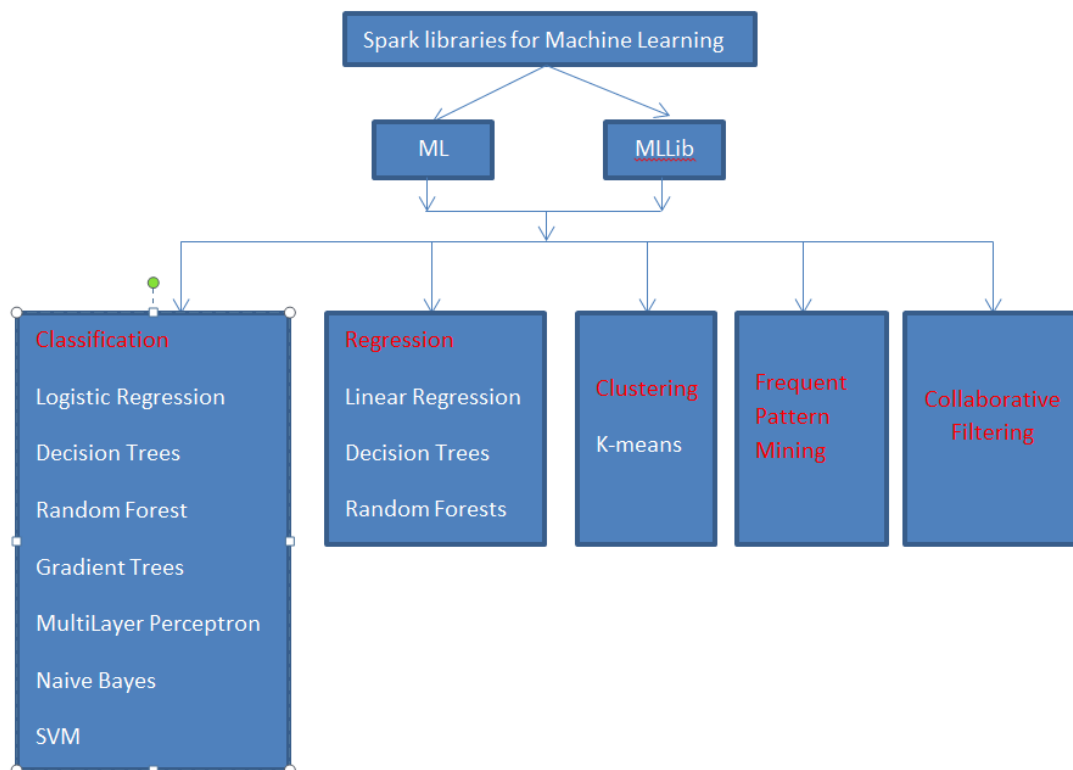


Fig. 1: Machine Learning Algorithms in Spark

2. LITERATURE REVIEW

Work done by various researchers to implement Machine Learning Techniques on Spark platform has been studied in this section. P. Hung *et al.* implemented Decision Tree algorithm using Pyspark ML and MLLib libraries on Breast Cancer dataset and the model provided the accuracy rate of 71% and 83 % respectively [1]. F Alarsan *et al.* implemented Random Forest and Gradient Boosted Trees on MIT BIH Arrhythmia dataset containing 205,146 records using MLLib and Scala language on local host and it was found that Random Forest performed superior over Gradient Boosted Trees [2]. W. Etaiwi *et al.* evaluated Naïve Bayes and Support Vector Machines on the dataset of Satindar Bank of Spain containing more than 14 million records using MLLib library and it was analyzed that Naïve Bayes overcomes Support Vector Machines in terms of precision, recall and F-Measure [3].

H. Sayed *et al.* compared performance of Decision Trees on both ML and MLLib package for Bank customer's dataset and it was found that in terms of testing time and accuracy, ML performed better while in terms of data transformations, MLLib took lesser time [4]. M. Kadampur and S. Riyacee introduced a tool named DLS Studio for applying deep learning model in cloud to classify image dataset and achieved ROC value of 99.77% [5]. K. Li *et al.* presented CMS, a container based continuous machine learning platform which simplified model training and deployment process with minimal human interference [6].

T. Craneiro *et al.* compared the performance of Deep learning applications on Google Colab, Distributed Hardware and Mainstream Workstations and it was concluded that Colab performed equivalent to dedicated hardware [7]. A framework has been presented by C. Tianshi *et al.* by integrating Jupyter notebook with Spark for scalable big data mining [8]. P. Gupta *et al.* evaluated 2 serverless variants: one with mapper and the other one with both mappers and reducers and the results proved that execution time of machine is sub-linear to the number of reducers called [9]. S. Santana *et al.* too deployed a scalable environment for data analysis task using Dockers [10].

Authors have also implemented LSTM i.e. Long-Short Term Memory Networks on Spark and

the results proved to be satisfactory. J. Zhang *et al.* implemented LSTM on cluster of 9 workstations and as per results, Cluster-Based LSTM performed superior over ordinary LSTM in terms of RMSE [11]. O. Aydin *et al.* implemented LSTM on Spark using Keras and Elephas library for distributed computing and the resulting model proved to be reliable [12]. S. Kumar *et al.* implemented LSTM and GRU upto 3 hidden layers for energy load forecasting using cluster of 7 machines and it was concluded that GRU performed better than LSTM [13]. It was also concluded that with the use of cluster, training time is 6 times faster. E. Huh *et al.* analyzed the performance of LSTM on containers and host environment with respect to CPU and GPU and as per results, training time of docker is less than local host [14].

From the review of work done by various researchers, it can be concluded that Spark performed satisfactory for implementing machine learning algorithms and execution time is considerably reduced for big databases. Machine Learning algorithms implemented by various researchers in Spark performed well in terms of both accuracy and training time.

3. EXPERIMENTAL SETUP

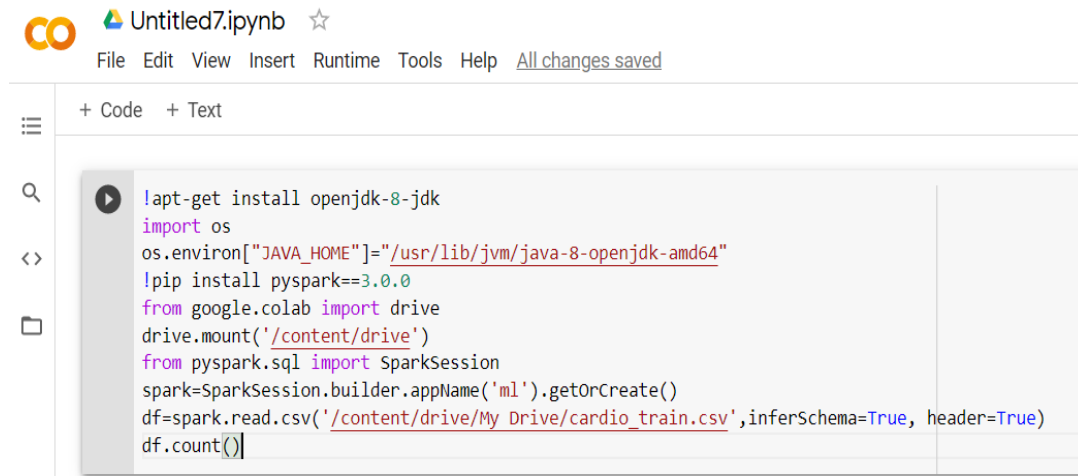
As discussed earlier in the previous section, Spark is a popular and efficient platform to perform learning from big data. Environment for spark can be setup via number of ways i.e. Azure Databricks, Amazon Elastic Map Reduce, Google Cloud, Dockers etc. In this section, Spark environment has been set up on local PC and on Google Colaboratory to implement Machine Learning algorithms.

Spark can be implemented on local machine by installing Java, Python, Winutils, Spark and Anaconda distribution (which uses IPython kernel at its backend) available freely on web. Python, Java and Scala are the programming languages used for Spark. Anaconda is a distribution of python and R languages which simplifies the task of package management and its deployment. Jupyter Notebook is a web application which supports Julia, Python and R programming language and allows user to create and share documents.

Google Colaboratory is an interactive environment developed by Google for writing and

IMPLEMENTING MACHINE LEARNING ALGORITHMS ON SPARK

executing python codes and can be run on CPU, GPU or TPU. Data from local PC can be uploaded to Google Drive and can then be used in Google Colab notebooks. Spark can also be run on Google Colab by installing all the necessary spark dependencies in Colab environment as shown in Figure 2. The major limitation of Colab is that it is available for only 12 hours per session.



```

!apt-get install openjdk-8-jdk
import os
os.environ["JAVA_HOME"]="/usr/lib/jvm/java-8-openjdk-amd64"
!pip install pyspark==3.0.0
from google.colab import drive
drive.mount('/content/drive')
from pyspark.sql import SparkSession
spark=SparkSession.builder.appName('ml').getOrCreate()
df=spark.read.csv('/content/drive/My Drive/cardio_train.csv',inferSchema=True, header=True)
df.count()

```

Fig. 2: Python code for creating SparkContext on Google Colab

4. IMPLEMENTATION

After setting up spark environment as specified in above section, the next step is to implement Machine Learning algorithms. Decision Trees, Random Forests and Gradient Boosted Trees have been implemented via Jupyter notebook on local PC (16GB RAM, 1TB of memory and 1.8 Ghz processing speed) and on Google Colaboratory and their relative performance in terms of accuracy and training time has been observed.

Heart Disease and Telecom Churn dataset containing 70,000 and 3,333 records respectively (available freely on Kaggle.com) has been used for implementation purpose [16, 17]. Input attributes for heart disease dataset i.e. Dataset1 are Age, Gender, Height, Weight, Cholesterol, High Blood Pressure, Low Blood Pressure, Cholesterol, Diabetes, whether a person smokes/drink and output is to classify whether a person is suffering from cardiac disease. For dataset 2 i.e. Telecom Churn dataset, input attributes are State, Account length, Area code,

International plan, Voice mail plan, Number of voice mail messages, Total day minutes, Total day calls, Total day charge, Total evening minutes, Total evening calls, Total eve charge, Total night minutes, Total night call etc. and output is to classify whether the customer will Churn or not.

Algorithms have been implemented on spark using Spark's ML library which has number of in-built functions to ease the task of implementation. To implement the algorithm, Transformer (accepts a dataframe as input and generates new dataframe by appending one or more column), Estimator (algorithm for training dataframe to create a model) and Pipelines (sequence of stages where each stage is either a transformer or estimator) are some basic concepts used.

A. DATA PREPARATION:

Input dataset contains a mix of numerical and categorical columns which needs to be handled cautiously while implementation so that categorical attributes may not be miss-treated as numerical ones. The categorical string values present in the input dataset first needs to be converted into an integer label via StringIndexer class which also stores the metadata of attributes. String values can too be retrieved back from the generated integer labels via IndexToString class. As there is no ordinal relation among these newly generated integer labels, One Hot Encoding of attributes via OneHotEncoder class is performed which converts the integer labels into binary vector as shown in Figure 3.

```
In [ ]: from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler

for column in Columns:
    indexers =StringIndexer(inputCol=column, outputCol=column+"_index")
    encoders = OneHotEncoder( inputCol=indexers.getOutputCol(), outputCol=column+"_encoded")
    stages+=[indexers,encoders]
```

Fig. 3: Python code for StringIndexer and OneHotEncoder

ML algorithm accepts input data in the form of a vector. So, the next step is to assemble all the input attributes i.e. numerical and the output of One Hot Encoder in a single vector column via VectorAssembler class (as shown in Figure 4). VectorAssembler class provides sparse vector as its output i.e. vector with huge number of zeroes as its output while ML algorithm accepts dense vector (as shown in Figure 6). Thus, sparse vector needs to be converted into dense vector which is then given as input to VectorIndexer class. VectorIndexer distinguish

between continuous and categorical values and also assign index to categorical values.

```
In [ ]: from pyspark.ml.feature import VectorAssembler  
  
assembler = VectorAssembler(inputCols=cols,outputCol="features")  
  
df1=assembler.transform(df)
```

Fig. 4: Python code for VectorAssembler class

B. MODEL TRAINING AND EVALUATION

The output given by VectorIndexer class is then given as input to machine learning algorithm. As shown in Figure 5, 'indexeddatafeatures', output column of VectorIndexer class is given as input to Decision Tree algorithm. For training the model, training and testing data is split in the ratio of 70:30 respectively. Fit function has been used to train the model on dataframe and to generate the prediction from the dataset transform function has been used.

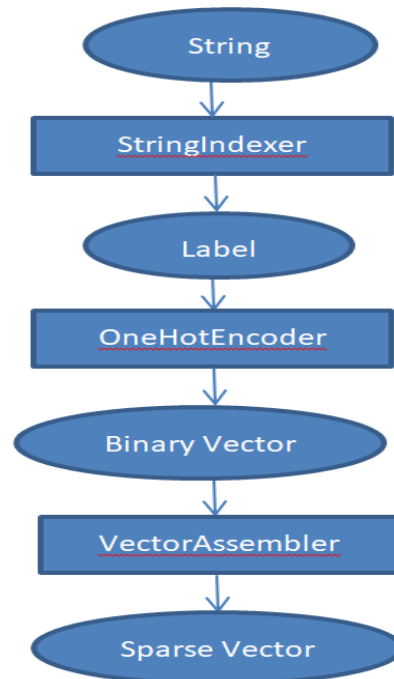


Fig. 5: Input to ML algorithm

```
In [ ]: from pyspark.ml.classification import DecisionTreeClassifier
(train_df, test_df) = df7.randomSplit([0.7, 0.3])
dt = DecisionTreeClassifier(labelCol="churn_index", featuresCol="indexedatafeatures")
```

Fig. 6: Implementation of Decision Tree Classifier

After training the data, MultiClassClassificationEvaluator (as shown in Figure 7) has been used to evaluate the accuracy of the model. Parameters of algorithm can then be fine-tuned to achieve maximum accuracy and minimum error.

```
In [ ]: from pyspark.ml.evaluation import MulticlassClassificationEvaluator
evaluator = MulticlassClassificationEvaluator(labelCol="churn_index", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test Error = %g " % (1.0 - accuracy))
```

Fig. 7: MultiClassClassificationEvaluator

5. EXPERIMENTAL RESULTS

Decision Trees, Random Forests and Gradient Boosted Trees have been implemented using Spark's ML library and performance of algorithm have been compared in terms of both accuracy and runtime. For dataset 1, Gradient Boosted Trees provides the most accurate results i.e. 73.62% while Decision Trees and Random Forests gave less accurate results i.e. 73.26% and 73.15% respectively on local PC as referred in Table1. Similar results have been observed for dataset 2 on local PC with accuracy value of 92.98%, 88.17% and 93.46% on Decision Tree, Random Forests and Gradient Boosted Trees respectively.

Table1: Comparison of ML algorithm on local host

	Dataset1		Dataset2	
	Accuracy	Runtime	Accuracy	Runtime
Decision Tree	73.2645	21.86758	92.9825	8.159213
Random Forest	73.1545	22.47601	88.168	11.37834
Gradient Boosted Trees	73.6278	25.01397	93.462	22.5124

Table2: Comparison of ML algorithm on Google Colab

Colab GPU	Dataset1		Dataset2	
	Accuracy	Runtime	Accuracy	Runtime
Decision Tree	72.5648	15.69723	92.0202	3.295617
Random Forest	71.9994	13.78304	89.9798	2.896695
Gradient Boosted Trees	72.7996	20.54381	95.0939	8.328558

On local PC, Decision Trees and Random Forests took lesser training time while Gradient Boosted Tree took the maximum training time on both the datasets. The experiments are then repeated on Google Colaboratory and similar results have been achieved in terms of accuracy of models while in terms of training time, algorithms took much lesser time on Colab GPU as mentioned in Table 2.

6. CONCLUSION AND FUTURE WORK

Apache Spark is the most popular framework to process big databases which has ML and MLlib library to implement machine learning algorithms. In this research work, Decision Trees, Random Forests and Gradient Boosted Trees have been implemented on Spark using ML Library it can be concluded that Gradient Boosted Tree is slowest among Decision Tree and Random Forest but provides the most accurate results. Algorithms have also been implemented on Google Colaboratory as well by selecting the GPU runtime and from the results it can be inferred that results are similar to the algorithms run on local machine in terms of accuracy but the run time of algorithm is considerably reduced on Google Colab.

In future, other machine learning algorithms can also be implemented using Spark's ML library and much bigger databases can be considered while implementation. Transfer learning and Deep Learning algorithms can also be implemented to further improve the accuracy of the model.

CONFLICT OF INTERESTS

The author(s) declare that there is no conflict of interests.

REFERENCES

- [1] P.D. Hung, T.D. Hanh, V.T. Diep, Breast cancer prediction using spark MLlib and ML packages, in: Proceedings of the 2018 5th International Conference on Bioinformatics Research and Applications, ACM, Hong Kong, 2018: pp. 52–59.
- [2] F.I. Alarsan, M. Younes, Analysis and classification of heart diseases using heartbeat features and machine learning algorithms, *J. Big Data*. 6 (2019), 81.
- [3] W. Etaiwi, M. Biltawi, G. Naymat, Evaluation of classification algorithms for banking customer's behavior under apache spark data processing system, *Procedia Computer Sci.* 113 (2017), 559–564.
- [4] H. Sayed, M. Abdel-Fattah, S. Kholief, Predicting potential banking customer churn using apache spark ml and MLlib packages: a comparative study, *Int. J. Adv. Computer Sci. Appl.* 9 (2018), 674–677.
- [5] M.A. Kadampur, S. Al Riyaae, Skin cancer detection: Applying a deep learning based model driven architecture in the cloud for classifying dermal cell images, *Inform. Med. Unlocked*. 18 (2020), 100282.
- [6] K. Li, N. Gui, CMS: A continuous machine-learning and serving platform for industrial big data, *Future Internet*. 12 (2020), 102.
- [7] T. Carneiro, R.V. Medeiros Da Nobrega, T. Nepomuceno, et al. Performance analysis of google colab as a tool for accelerating deep learning applications, *IEEE Access*. 6 (2018), 61677–61685.
- [8] C. Tianshi, J. Wei, Scalable and cooperative big data mining platform design for smart grid, in: 2016 China International Conference on Electricity Distribution (CICED), IEEE, Xi'an, China, 2016: pp. 1–5.
- [9] P. Gupta, S. Addala, Experimental evaluation of serverless functions, <https://g31pranjal.github.io/assets/serverless-report.pdf>.
- [10] S. Martín-Santana, C.J. Pérez-González, M. Colebrook, J.L. Roda-García, P. González-Yanes, Deploying a scalable data science environment using docker, in: F.P. García Márquez, B. Lev (Eds.), *Data Science and Digital Business*, Springer International Publishing, Cham, 2019: pp. 121–146.
- [11] J. Zhang, F. Chen, Q. Shen, Cluster-based LSTM network for short-term passenger flow forecasting in urban rail transit, *IEEE Access*. 7 (2019), 147653–147671.
- [12] O. Aydin, S. Guldamlasioglu, Using LSTM networks to predict engine condition on large scale data processing framework, in: 2017 4th International Conference on Electrical and Electronic Engineering (ICEEE), IEEE,

IMPLEMENTING MACHINE LEARNING ALGORITHMS ON SPARK

Ankara, Turkey, 2017: pp. 281–285.

- [13] S. Kumar, L. Hussain, S. Banarjee, M. Reza, Energy load forecasting using deep learning approach-LSTM and GRU in spark cluster, in: 2018 Fifth International Conference on Emerging Applications of Information Technology (EAIT), IEEE, Kolkata, 2018: pp. 1–4.
- [14] T.D.T. Nguyen, E.-N. Huh, J.H. Park, et al. Performance analysis of data parallelism technique in machine learning for human activity recognition using LSTM, in: 2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), IEEE, Sydney, Australia, 2019: pp. 387–391.
- [15] Kaggle, Cardiovascular Classification, <https://www.kaggle.com/sulianova/cardiovascular-disease-dataset>, accessed January 1, 2019.
- [16] Kaggle, mlcourse.ai, Open Machine Learning Course by OpenDataScience, https://www.kaggle.com/kashnitsky/mlcourse?select=telecom_churn.csv, accessed June 18, 2018.