Project Report: PySpark Linear Regression Evaluation Function

Objective

This project aims to revolutionize the linear regression model evaluation process within PySpark environments by developing a specialized function. The primary goal is to address challenges associated with manual implementation errors and time-intensive evaluation tasks faced by data scientists and machine learning practitioners. The function provides a user-friendly interface for the assessment and comparison of linear regression models, offering essential performance metrics such as R-squared, mean squared error (MSE), and root mean squared error (RMSE). This comprehensive toolkit not only streamlines the evaluation process but also contributes to informed decision-making during model selection. By encapsulating the evaluation logic within the PySpark framework, the project enhances efficiency and facilitates model comparison, empowering users to iteratively experiment with different features and algorithms. The scalable nature of the solution ensures its applicability to large-scale datasets, while the modular design encourages future extensions and collaborative enhancements, fostering a dynamic environment for continuous improvement in PySpark-based predictive modeling practices.

Implementation

1. Overview of Linear Regression Function

The PySpark-based linear regression evaluation function leverages the Linear Regression module to automate the process of training, prediction, and evaluation. It allows developers to assess the effectiveness of different linear regression models without manual implementation, saving time and effort.

Fig.1 and Fig. 2 illustrates the function.

Fig.1

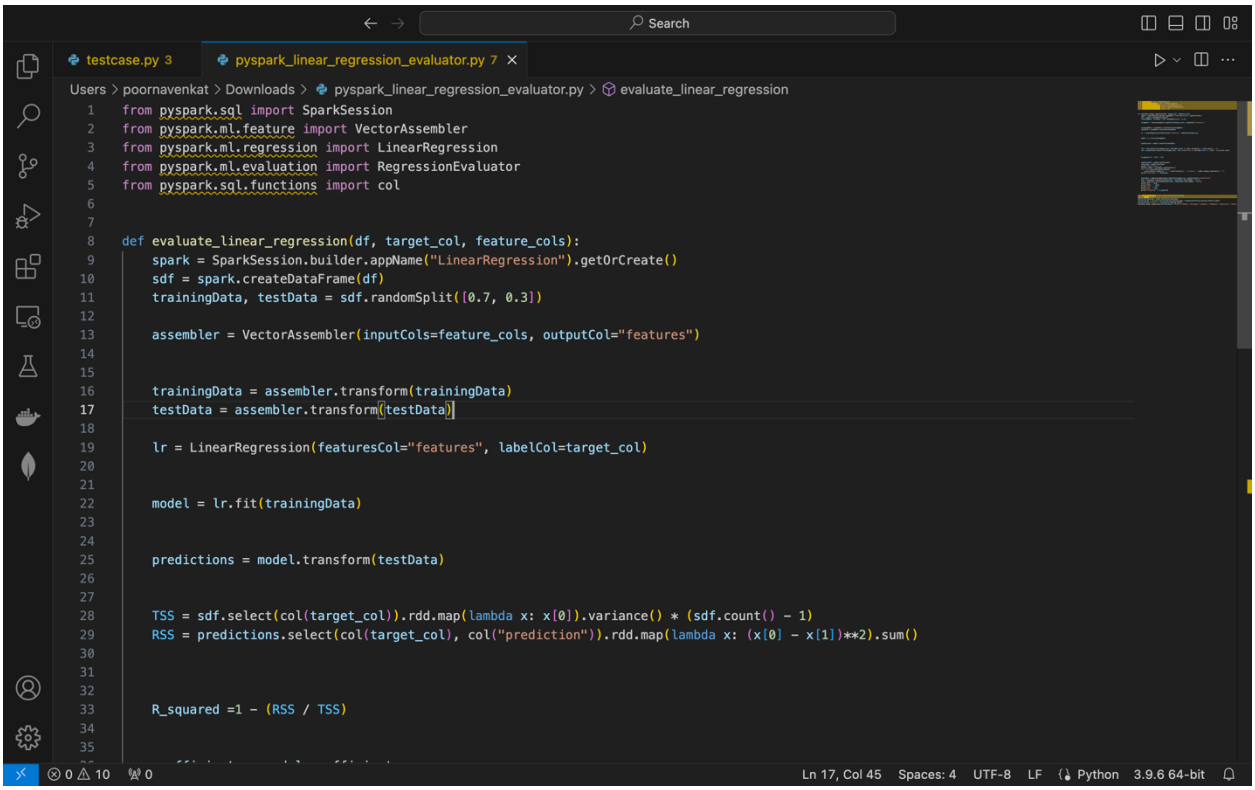
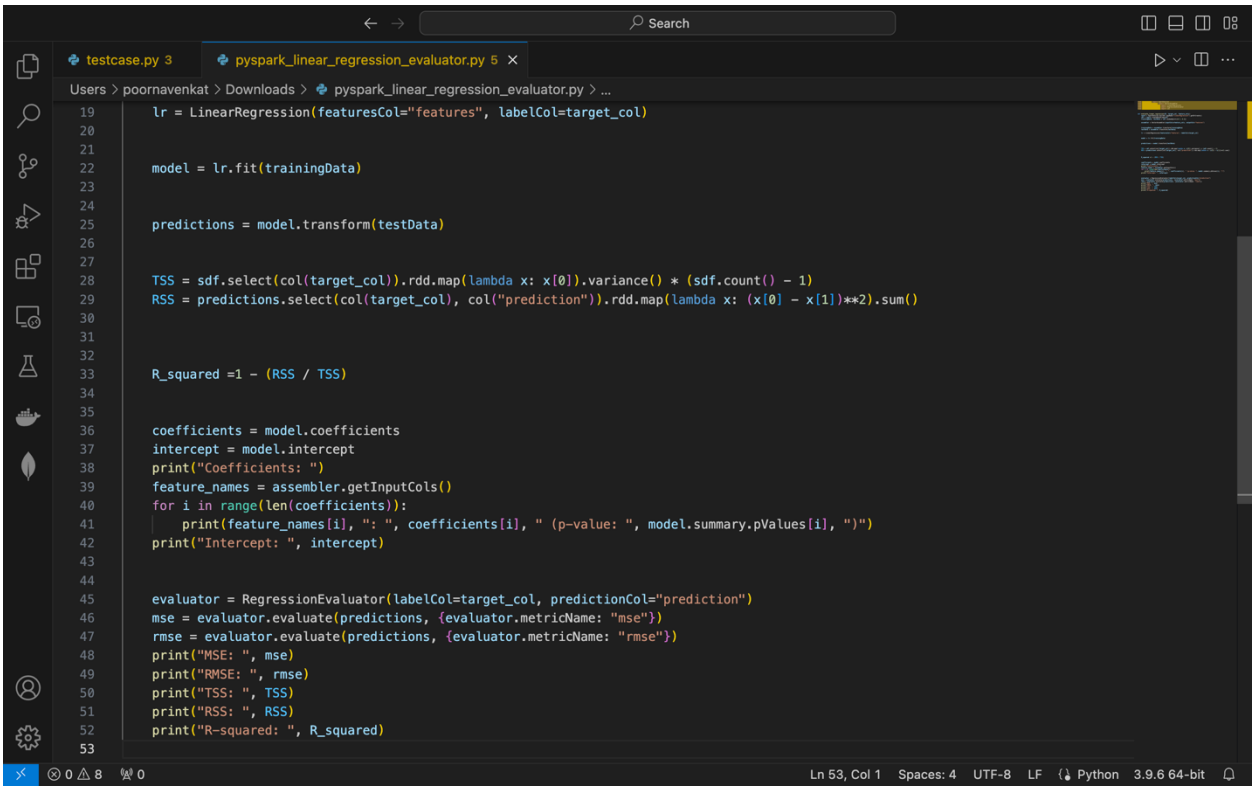

Search
testcase.py 3
pyspark_linear_regression_evaluator.py 7
Users > poornavenkat > Downloads > pyspark_linear_regression_evaluator.py > evaluate_linear_regression
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.sql.functions import col
def evaluate_linear_regression(df, target_col, feature_cols):
spark = SparkSession.builder.appName("LinearRegression").getOrCreate()
sdf = spark.createDataFrame(df)
trainingData, testData = sdf.randomSplit([0.7, 0.3])
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
trainingData = assembler.transform(trainingData)
testData = assembler.transform(testData)
lr = LinearRegression(featuresCol="features", labelCol=target_col)
model = lr.fit(trainingData)
predictions = model.transform(testData)
TSS = sdf.select(col(target_col)).rdd.map(lambda x: x[0]).variance() * (sdf.count() - 1)
RSS = predictions.select(col(target_col), col("prediction")).rdd.map(lambda x: (x[0] - x[1])**2).sum()
R_squared =1 - (RSS / TSS)
Ln 17, Col 45
Spaces: 4
UTF-8
LF
Python
3.9.6 64-bit

Fig.2



2. Function Workflow

The function follows a structured workflow, including data splitting, model initialization, pipeline creation, model training, prediction, and performance evaluation. This systematic approach ensures a comprehensive analysis of linear regression models.

3. Performance Metrics

Key performance metrics, such as

1. R-squared
2. MSE
3. RMSE
4. TSS
5. RSS and
6. R-square are returned by the function. These metrics offer valuable insights into

The quality and accuracy of the linear regression models.

Integration and Portability.

4. Module Integration

The PySpark-based linear regression evaluation function is designed to be modular, allowing seamless integration into any PySpark environment. It can be treated as a private PySpark module, promoting code reusability and maintaining a clean and organized structure.

We import the necessary modules, including SparkSession and the evaluate_linear_regression function from the custom module (pyspark_linear_regression_evaluator.py).

```
1  from pyspark_linear_regression_evaluator import evaluate_linear_regression
2
```

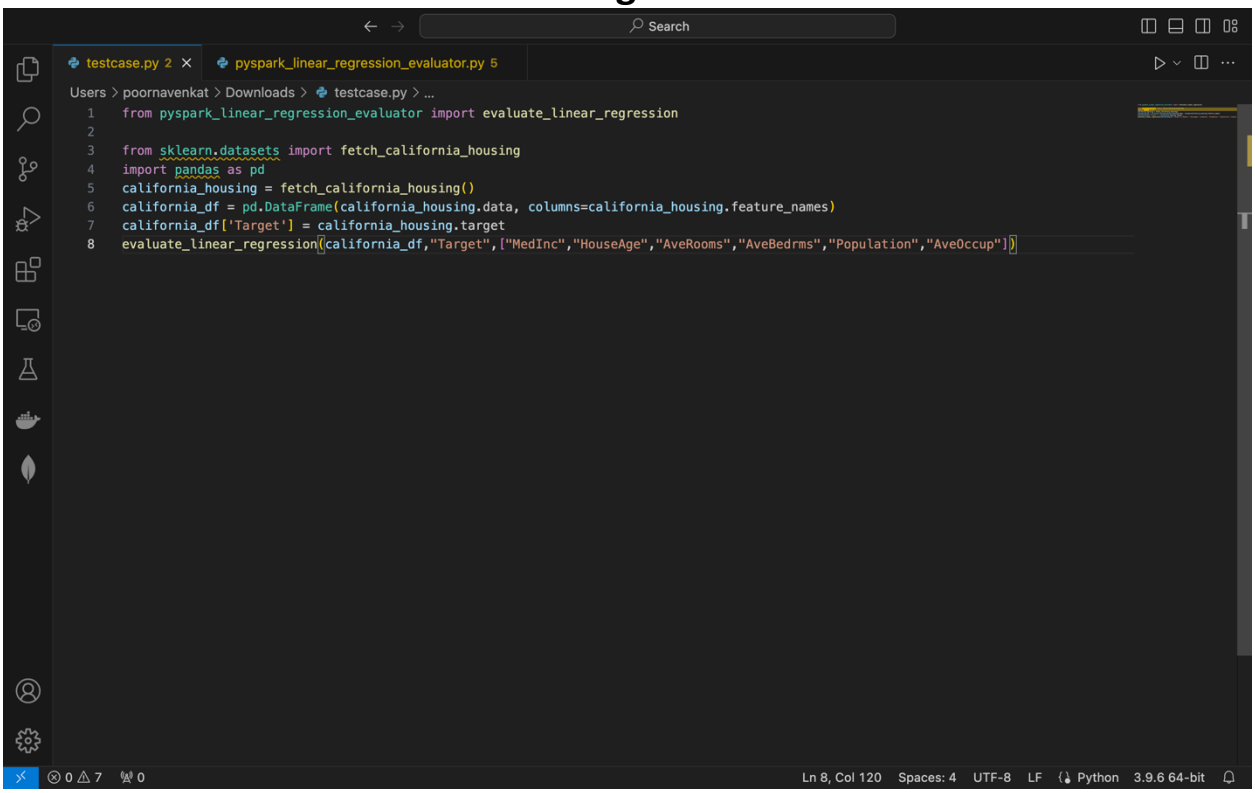
5.Example Usage in a PySpark Environment

The function can be easily integrated into a PySpark script, as demonstrated in the example usage. By importing the function from the private module, developers can efficiently evaluate linear regression models within their PySpark projects.

We import the necessary modules, including SparkSession and the evaluate_linear_regression function from the custom module (pyspark_linear_regression_evaluator.py). and created a testcase file with dataset fetch_california_housing importing from sklearn.datasets.

The testcase file is shown in Fig.3

Fig.3



And outputs after running the file is shown in Fig.4 and Fig.5

Fig.4

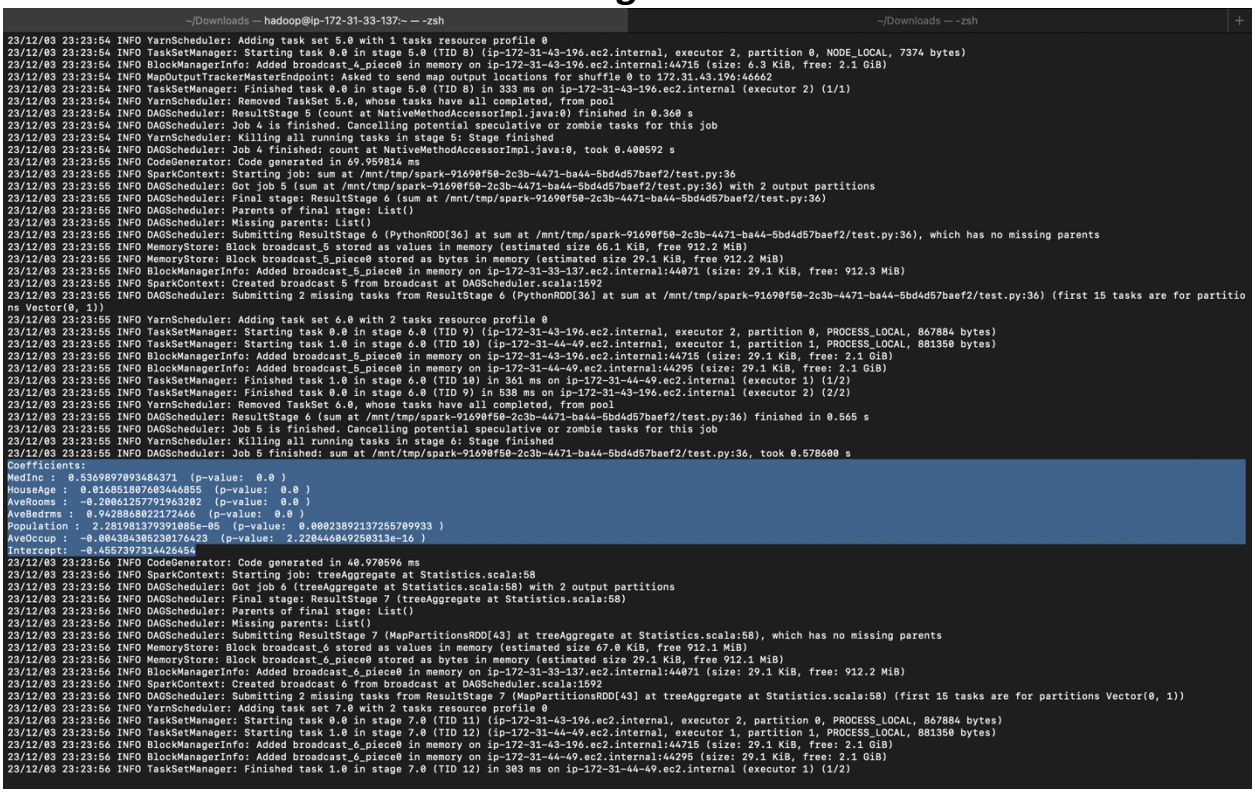
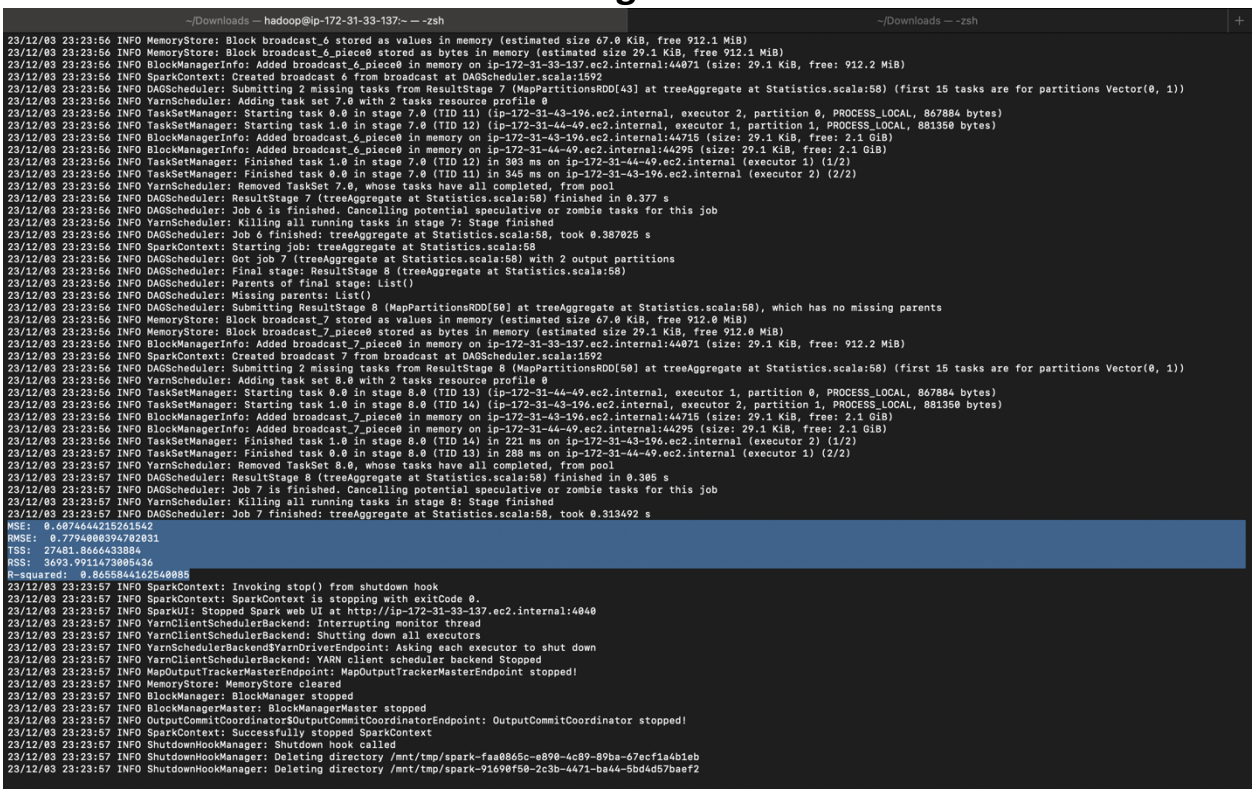

~/Downloads — hadoop@ip-172-31-33-137:~ — -zsh
~/Downloads — -zsh
23/12/03 23:23:54 INFO YarnScheduler: Adding task set 5.0 with 1 tasks resource profile 0
23/12/03 23:23:54 INFO TaskSetManager: Starting task 0.0 in stage 5.0 (TID 8) (ip-172-31-43-196.ec2.internal, executor 2, partition 0, NODE_LOCAL, 7374 bytes)
23/12/03 23:23:54 INFO BlockManagerInfo: Added broadcast_4_piece0 in memory on ip-172-31-43-196.ec2.internal:44715 (size: 6.3 KiB, free: 2.1 GiB)
23/12/03 23:23:54 INFO MapOutputTrackerMasterEndpoint: Asked to send map output locations for shuffle 0 to 172.31.43.196:46662
23/12/03 23:23:54 INFO TaskSetManager: Finished task 0.0 in stage 5.0 (TID 8) in 333 ms on ip-172-31-43-196.ec2.internal (executor 2) (1/1)
23/12/03 23:23:54 INFO YarnScheduler: Removed TaskSet 5.0, whose tasks have all completed, from pool
23/12/03 23:23:54 INFO DAGScheduler: ResultStage 5 (count at NativeMethodAccessorImpl.java:0) finished in 0.360 s
23/12/03 23:23:54 INFO DAGScheduler: Job 4 is finished. Cancelling potential speculative or zombie tasks for this job
23/12/03 23:23:54 INFO YarnScheduler: Killing all running tasks in stage 5: Stage finished
23/12/03 23:23:54 INFO DAGScheduler: Job 4 finished: count at NativeMethodAccessorImpl.java:0, took 0.400592 s
23/12/03 23:23:55 INFO CodeGenerator: Code generated in 69.959814 ms
23/12/03 23:23:55 INFO SparkContext: Starting job: sum at /mnt/tmp/spark-91690f50-2c3b-4471-ba44-5bd4d57baef2/test.py:36
23/12/03 23:23:55 INFO DAGScheduler: Got job 5 (sum at /mnt/tmp/spark-91690f50-2c3b-4471-ba44-5bd4d57baef2/test.py:36) with 2 output partitions
23/12/03 23:23:55 INFO DAGScheduler: Final stage: ResultStage 6 (sum at /mnt/tmp/spark-91690f50-2c3b-4471-ba44-5bd4d57baef2/test.py:36)
23/12/03 23:23:55 INFO DAGScheduler: Parents of final stage: List()
23/12/03 23:23:55 INFO DAGScheduler: Missing parents: List()
23/12/03 23:23:55 INFO DAGScheduler: Submitting ResultStage 6 (PythonRDD[36] at sum at /mnt/tmp/spark-91690f50-2c3b-4471-ba44-5bd4d57baef2/test.py:36), which has no missing parents
23/12/03 23:23:55 INFO MemoryStore: Block broadcast_5 stored as values in memory (estimated size 65.1 KiB, free 912.2 MiB)
23/12/03 23:23:55 INFO MemoryStore: Block broadcast_5_piece0 stored as bytes in memory (estimated size 29.1 KiB, free 912.2 MiB)
23/12/03 23:23:55 INFO BlockManagerInfo: Added broadcast_5_piece0 in memory on ip-172-31-33-137.ec2.internal:44071 (size: 29.1 KiB, free: 912.3 MiB)
23/12/03 23:23:55 INFO SparkContext: Created broadcast 5 from broadcast at DAGScheduler.scala:1592
23/12/03 23:23:55 INFO DAGScheduler: Submitting 2 missing tasks from ResultStage 6 (PythonRDD[36] at sum at /mnt/tmp/spark-91690f50-2c3b-4471-ba44-5bd4d57baef2/test.py:36) (first 15 tasks are for partitio
ns Vector(0, 1))
23/12/03 23:23:55 INFO YarnScheduler: Adding task set 6.0 with 2 tasks resource profile 0
23/12/03 23:23:55 INFO TaskSetManager: Starting task 0.0 in stage 6.0 (TID 9) (ip-172-31-43-196.ec2.internal, executor 2, partition 0, PROCESS_LOCAL, 867884 bytes)
23/12/03 23:23:55 INFO TaskSetManager: Starting task 1.0 in stage 6.0 (TID 10) (ip-172-31-44-49.ec2.internal, executor 1, partition 1, PROCESS_LOCAL, 881350 bytes)
23/12/03 23:23:55 INFO BlockManagerInfo: Added broadcast_5_piece0 in memory on ip-172-31-43-196.ec2.internal:44715 (size: 29.1 KiB, free: 2.1 GiB)
23/12/03 23:23:55 INFO BlockManagerInfo: Added broadcast_5_piece0 in memory on ip-172-31-44-49.ec2.internal:44295 (size: 29.1 KiB, free: 2.1 GiB)
23/12/03 23:23:55 INFO TaskSetManager: Finished task 1.0 in stage 6.0 (TID 10) in 361 ms on ip-172-31-44-49.ec2.internal (executor 1) (1/2)
23/12/03 23:23:55 INFO TaskSetManager: Finished task 0.0 in stage 6.0 (TID 9) in 538 ms on ip-172-31-43-196.ec2.internal (executor 2) (2/2)
23/12/03 23:23:55 INFO YarnScheduler: Removed TaskSet 6.0, whose tasks have all completed, from pool
23/12/03 23:23:55 INFO DAGScheduler: ResultStage 6 (sum at /mnt/tmp/spark-91690f50-2c3b-4471-ba44-5bd4d57baef2/test.py:36) finished in 0.565 s
23/12/03 23:23:55 INFO DAGScheduler: Job 5 is finished. Cancelling potential speculative or zombie tasks for this job
23/12/03 23:23:55 INFO YarnScheduler: Killing all running tasks in stage 6: Stage finished
23/12/03 23:23:55 INFO DAGScheduler: Job 5 finished: sum at /mnt/tmp/spark-91690f50-2c3b-4471-ba44-5bd4d57baef2/test.py:36, took 0.578600 s
Coefficients:
MedInc : 0.5369897093484371 (p-value: 0.0)
HouseAge : 0.01685180760346855 (p-value: 0.0)
AveRooms : -0.2006125779196320 2 (p-value: 0.0)
AveBedrms : 0.9428868022172466 (p-value: 0.0)
Population : 2.281981379391085e-05 (p-value: 0.00023892137255709933)
AveOccup : -0.004384305230176423 (p-value: 2.220446049250313e-16)
Intercept: -0.4557397314426454
23/12/03 23:23:56 INFO CodeGenerator: Code generated in 40.970596 ms
23/12/03 23:23:56 INFO SparkContext: Starting job: treeAggregate at Statistics.scala:58
23/12/03 23:23:56 INFO DAGScheduler: Got job 6 (treeAggregate at Statistics.scala:58) with 2 output partitions
23/12/03 23:23:56 INFO DAGScheduler: Final stage: ResultStage 7 (treeAggregate at Statistics.scala:58)
23/12/03 23:23:56 INFO DAGScheduler: Parents of final stage: List()
23/12/03 23:23:56 INFO DAGScheduler: Missing parents: List()
23/12/03 23:23:56 INFO DAGScheduler: Submitting ResultStage 7 (MapPartitionsRDD[43] at treeAggregate at Statistics.scala:58), which has no missing parents
23/12/03 23:23:56 INFO MemoryStore: Block broadcast_6 stored as values in memory (estimated size 67.0 KiB, free 912.1 MiB)
23/12/03 23:23:56 INFO MemoryStore: Block broadcast_6_piece0 stored as bytes in memory (estimated size 29.1 KiB, free 912.1 MiB)
23/12/03 23:23:56 INFO BlockManagerInfo: Added broadcast_6_piece0 in memory on ip-172-31-33-137.ec2.internal:44071 (size: 29.1 KiB, free: 912.2 MiB)
23/12/03 23:23:56 INFO SparkContext: Created broadcast 6 from broadcast at DAGScheduler.scala:1592
23/12/03 23:23:56 INFO DAGScheduler: Submitting 2 missing tasks from ResultStage 7 (MapPartitionsRDD[43] at treeAggregate at Statistics.scala:58) (first 15 tasks are for partitions Vector(0, 1))
23/12/03 23:23:56 INFO YarnScheduler: Adding task set 7.0 with 2 tasks resource profile 0
23/12/03 23:23:56 INFO TaskSetManager: Starting task 0.0 in stage 7.0 (TID 11) (ip-172-31-43-196.ec2.internal, executor 2, partition 0, PROCESS_LOCAL, 867884 bytes)
23/12/03 23:23:56 INFO TaskSetManager: Starting task 1.0 in stage 7.0 (TID 12) (ip-172-31-44-49.ec2.internal, executor 1, partition 1, PROCESS_LOCAL, 881350 bytes)
23/12/03 23:23:56 INFO BlockManagerInfo: Added broadcast_6_piece0 in memory on ip-172-31-43-196.ec2.internal:44715 (size: 29.1 KiB, free: 2.1 GiB)
23/12/03 23:23:56 INFO BlockManagerInfo: Added broadcast_6_piece0 in memory on ip-172-31-44-49.ec2.internal:44295 (size: 29.1 KiB, free: 2.1 GiB)
23/12/03 23:23:56 INFO TaskSetManager: Finished task 1.0 in stage 7.0 (TID 12) in 303 ms on ip-172-31-44-49.ec2.internal (executor 1) (1/2)

Fig.5



My github link:

https://github.com/chaparalatharun/big-data-project/issues

Conclusion

The PySpark-based linear regression evaluation function offers a powerful and flexible tool for assessing and comparing linear regression models in a PySpark environment. Its modular design, seamless integration, and provision of key performance metrics contribute to time savings, code efficiency, and enhanced decision-making in model development.

This consolidated report provides a comprehensive overview of the PySpark-based linear regression evaluation function, covering its implementation details, key features, integration into PySpark workflows, and example usage.