**EXPERIMENT 8**

**AIM:** Execute any two ML Algorithms using Apache Spark MLlib and compare the results.

**THEORY:**

**Spark MLlib:**

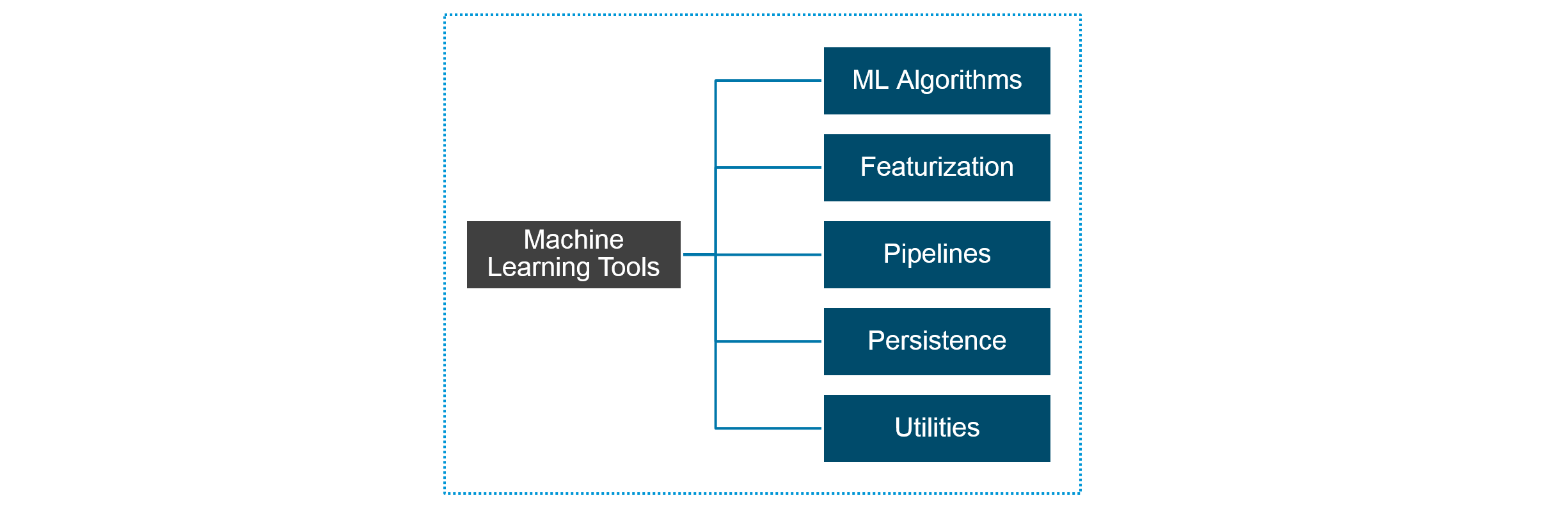
Spark MLlib is Apache Spark’s Machine Learning component. MLlib consists of popular algorithms and utilities. MLlib in Spark is a scalable Machine learning library that discusses both high-quality algorithm and high speed. The machine learning algorithms like regression, classification, clustering, pattern mining, and collaborative filtering. Lower level machine learning primitives like generic gradient descent optimization algorithm are also present in MLlib.

spark.mllib contains the original API built on top of RDDs. It is currently in maintenance mode.

spark.ml provides higher level API built on top of DataFrames for constructing ML pipelines. spark.ml is the primary Machine Learning API for Spark at the moment.

**Spark MLlib Tools :**

* **ML Algorithms:** ML Algorithms form the core of MLlib. These include common learning algorithms such as classification, regression, clustering and collaborative filtering.
* **Featurization:** Featurization includes feature extraction, transformation, dimensionality reduction and selection.
* **Pipelines:** Pipelines provide tools for constructing, evaluating and tuning ML Pipelines.
* **Persistence:** Persistence helps in saving and loading algorithms, models and Pipelines.
* **Utilities:** Utilities for linear algebra, statistics and data handling.



**MLlib Algorithms:**

The popular algorithms and utilities in Spark MLlib are:

* Basic Statistics
* Regression
* Classification
* Recommendation System
* Clustering
* Dimensionality Reduction
* Feature Extraction
* Optimization

**CODE:**

# install pyspark

!pip install pyspark

# initialise session

from pyspark.context import SparkContext

from pyspark.sql.session import SparkSession

from pyspark.sql import SQLContext

sc = SparkContext('local')

spark = SparkSession(sc)

sqlContext = SQLContext(sc)

1. **Regeression**

df = sqlContext.read.format('com.databricks.spark.csv').options(header = 'true', inferschema= 'true').load('healthcare-dataset-stroke-data.csv')

df.take(1)

pd\_df = df.toPandas()

print(pd\_df)

pd\_df.dtypes

df.cache()

df.printSchema()

df.cache()

df.printSchema()

df.head()

from pyspark.ml.feature import VectorAssembler

vectorAssembler = VectorAssembler(inputCols = ['age', 'avg\_glucose\_level', 'hypertension', 'heart\_disease', 'stroke'], outputCol = 'features')

tdf = vectorAssembler.transform(df)

print(tdf)

tdf = tdf.select(['features', 'stroke'])

tdf.show(3)

splits = tdf.randomSplit([0.7, 0.3])

train\_df = splits[0]

test\_df = splits[1]

from pyspark.ml.regression import LinearRegression

lr = LinearRegression(featuresCol = 'features', labelCol='stroke', maxIter=10, regParam=0.3, elasticNetParam=0.8)

lr\_model = lr.fit(train\_df)

print("Coefficients: " + str(lr\_model.coefficients))

print("Intercept: " + str(lr\_model.intercept))

trainingSummary = lr\_model.summary

print("RMSE: %f" % trainingSummary.rootMeanSquaredError)

print("r2: %f" % trainingSummary.r2)

1. **Classification**

df = sqlContext.read.format('com.databricks.spark.csv').options(header = 'true', inferschema = 'true' ).load( 'salary.csv' )

df.take(1)

df.cache()

df.printSchema()

df.dtypes

from pyspark.ml.feature import StringIndexer

indexer0 = StringIndexer(inputCol="workclass", outputCol="workclassEncoded")

indexed0 = indexer0.fit(df).transform(df)

indexer1 = StringIndexer(inputCol="education", outputCol="educationEncoded")

indexed1 = indexer1.fit(indexed0).transform(indexed0)

indexer2 = StringIndexer(inputCol="occupation", outputCol="occupationEncoded")

indexed2 = indexer2.fit(indexed1).transform(indexed1)

indexer3 = StringIndexer(inputCol="sex", outputCol="sexEncoded")

indexed3 = indexer3.fit(indexed2).transform(indexed2)

indexer4 = StringIndexer(inputCol="native-country", outputCol="countryEncoded")

indexed4 = indexer4.fit(indexed3).transform(indexed3)

indexer5 = StringIndexer(inputCol="salary", outputCol="salaryEncoded")

indexed5 = indexer5.fit(indexed4).transform(indexed4)

indexed5.show()

from pyspark.ml.feature import VectorAssembler

vectorAssembler = VectorAssembler(inputCols = ['age', 'workclassEncoded', 'educationEncoded', 'occupationEncoded', 'sexEncoded', 'countryEncoded', 'education-num', 'capital-gain', 'hours-per-week', 'capital-loss'], outputCol = 'features')

tdf = vectorAssembler.transform(indexed5)

print(tdf)

tdf = tdf.select(['features', 'salaryEncoded'])

tdf.show(3)

train, test = tdf.randomSplit([0.7, 0.3], seed = 2018)

print("Training Dataset Count: " + str(train.count()))

print("Test Dataset Count: " + str(test.count()))

from pyspark.ml.classification import LogisticRegression

from pyspark.ml.classification import LinearSVC

lsvc = LinearSVC(featuresCol = 'features', labelCol = 'salaryEncoded', maxIter=10, regParam=0.1)

lr = LogisticRegression(featuresCol = 'features', labelCol = 'salaryEncoded', maxIter=10)

lsvcModel = lsvc.fit(train)

lrModel = lr.fit(train)

print("Coefficients: " + str(lsvcModel.coefficients))

print("Intercept: " + str(lsvcModel.intercept))

import matplotlib.pyplot as plt

trainingSummary = lrModel.summary

roc = trainingSummary.roc.toPandas()

plt.plot(roc['FPR'],roc['TPR'])

plt.ylabel('False Positive Rate')

plt.xlabel('True Positive Rate')

plt.title('ROC Curve')

plt.show()

print('Training set areaUnderROC: ' + str(trainingSummary.areaUnderROC))

**OUTPUT:**

1. **Regression**

Calendar

Description automatically generated with medium confidenceA picture containing table

Description automatically generatedText

Description automatically generatedGraphical user interface, text

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generated

**Final Predictions**Text

Description automatically generatedGraphical user interface, text, application

Description automatically generated

1. **Classification**

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

**Final Predictions**

Text

Description automatically generated

Chart

Description automatically generated

**CONCLUSION:**

Thus in the experiment, we explored and learnt about pySpark’s machine learning library – MLlib which supports Basic Statistics, Regression, Classification, Recommendation System, Clustering and many more. We implemented Linear Regression algorithm on stroke healthcare dataset having an RSME of 0.213 and an R2 of 0.0 suggesting overfitting. We also implemented Classification on salary dataset giving us an AUC score of 0.847 suggesting a good fit. Thus, in this experimented we have implemented 2 ML algorithms supported by MLlib. The outputs were observed and attached above.