BIG DATA AND LARGE SCALE COMPUTING

Name: Abhishek harsh

Roll :2021BCS0036

**Code :**

import org.apache.spark.sql.SparkSession

import org.apache.spark.ml.feature.VectorAssembler

import org.apache.spark.ml.classification.\_

import org.apache.spark.ml.evaluation.{MulticlassClassificationEvaluator, BinaryClassificationEvaluator}

import org.apache.spark.sql.functions.\_

object ChildSleepQualityClassification {

def main(args: Array[String]): Unit = {

// Initialize SparkSession

val spark = SparkSession.builder()

.appName("ChildSleepQualityClassification")

.config("spark.master", "local[\*]")

.getOrCreate()

// Load dataset

val df = spark.read.option("header", "true").option("inferSchema", "true").csv("student\_sleep\_patterns.csv")

df.show(5)

df.printSchema()

// Assemble features

val featureColumns = Array("Age", "Sleep\_Duration", "Sleep\_Quality", "Screen\_Time", "Physical\_Activity")

val assembler = new VectorAssembler().setInputCols(featureColumns).setOutputCol("features")

val assembledData = assembler.transform(df)

// Label transformation (binary classification)

val remappedData = assembledData.withColumn("label", when(col("Sleep\_Duration") >= 6 && col("Sleep\_Duration") <= 8, 1).otherwise(0))

val finalData = remappedData.select(col("label"), col("features"))

val Array(trainingData, testData) = finalData.randomSplit(Array(0.7, 0.3), seed = 1234)

// Models

val models = Seq(

new DecisionTreeClassifier().setLabelCol("label").setFeaturesCol("features"),

new LogisticRegression().setLabelCol("label").setFeaturesCol("features"),

new GBTClassifier().setLabelCol("label").setFeaturesCol("features").setMaxIter(10),

new RandomForestClassifier().setLabelCol("label").setFeaturesCol("features").setNumTrees(10),

new NaiveBayes().setLabelCol("label").setFeaturesCol("features"),

new LinearSVC().setLabelCol("label").setFeaturesCol("features").setMaxIter(10)

)

// Evaluators

val evaluatorAccuracy = new MulticlassClassificationEvaluator().setLabelCol("label").setPredictionCol("prediction").setMetricName("accuracy")

val evaluatorPrecision = new MulticlassClassificationEvaluator().setLabelCol("label").setPredictionCol("prediction").setMetricName("weightedPrecision")

val evaluatorRecall = new MulticlassClassificationEvaluator().setLabelCol("label").setPredictionCol("prediction").setMetricName("weightedRecall")

val evaluatorF1 = new MulticlassClassificationEvaluator().setLabelCol("label").setPredictionCol("prediction").setMetricName("f1")

val evaluatorROC = new BinaryClassificationEvaluator().setLabelCol("label").setRawPredictionCol("prediction").setMetricName("areaUnderROC")

// Train, predict, and evaluate each model

models.zipWithIndex.foreach { case (model, index) =>

val modelName = model.getClass.getSimpleName

println(s"\nTraining $modelName...")

val trainedModel = model.fit(trainingData)

val predictions = trainedModel.transform(testData)

// Confusion matrix

println(s"Confusion matrix for $modelName:")

predictions.groupBy("label", "prediction").count().show()

// Metrics

val accuracy = evaluatorAccuracy.evaluate(predictions)

val precision = evaluatorPrecision.evaluate(predictions)

val recall = evaluatorRecall.evaluate(predictions)

val f1Score = evaluatorF1.evaluate(predictions)

val rocAuc = evaluatorROC.evaluate(predictions)

println(s"Metrics for $modelName:")

println(s"Accuracy: $accuracy")

println(s"Precision: $precision")

println(s"Recall: $recall")

println(s"F1 Score: $f1Score")

println(s"ROC AUC: $rocAuc")

}

// Stop SparkSession

spark.stop()

}

}