package com.vinspark.frauddetection

import org.apache.spark.ml.Pipeline

import org.apache.spark.ml.classification.DecisionTreeClassifier

import org.apache.spark.ml.classification.DecisionTreeClassificationModel

import org.apache.spark.ml.feature.{StringIndexer, IndexToString, VectorIndexer}

import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator

import org.apache.spark.mllib.util.MLUtils

import org.apache.spark.SparkContext.\_

import org.apache.spark.sql.\_

import org.apache.log4j.\_

import scala.io.Source

import java.nio.charset.CodingErrorAction

import scala.io.Codec

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

import org.apache.spark.sql.types

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

import org.apache.spark.ml.feature.IndexToString

import org.apache.spark.ml.attribute.Attribute

import org.apache.spark.sql.Column

object FraudDet\_ML2 extends Serializable {

def main(args:Array[String]){

Logger.getLogger("org").setLevel(Level.ERROR)

val spark=SparkSession

.builder()

.appName("FraudDetection")

.master("local[\*]")

.config("spark.sql.warehouse.dir","file:///C:/temp")

.getOrCreate()

import spark.sqlContext.implicits.\_

var df=spark.read.format("csv").option("header", "true").option("mode", "DROPMALFORMED").option("inferSchema", "true").load("../PS\_20174392719\_1491204439457\_log.csv")

df= df.withColumn("orgDiff", col("newbalanceOrig") - col("oldbalanceOrg")).withColumn("destDiff", col("newbalanceDest") - col("oldbalanceDest"))

// df.createOrReplaceTempView("transaction")

df= df.withColumn("label",

when((col("oldbalanceOrg") <=56900 && col("type")=="TRANSFER" && col("newbalanceDest") <= 105)

||(col("oldbalanceOrg") >56900 && col("newbalanceOrig")<=12)

||(col("oldbalanceOrg") >56900 && col("newbalanceOrig")>12 && col("amount")>1160000),

1)

.otherwise(0)

)

val stringColumns = Array("type", "amount", "oldbalanceOrg", "newbalanceOrig", "oldbalanceDest", "newbalanceDest", "orgDiff", "destDiff")

val stringColumns\_index = stringColumns.map(c => s"${c}\_index")

val index\_transformers = stringColumns.map(

cname => new StringIndexer()

.setInputCol(cname)

.setOutputCol(s"${cname}\_index")

)

// val indexer = new StringIndexer().setInputCol("type").setOutputCol("typeIndexed")

// val indexed = indexer.fit(df).transform(df)

/\* val labelIndexer = new StringIndexer()

.setInputCol("label")

.setOutputCol("indexedLabel")

.fit(df)\*/

// Automatically identify categorical features, and index them.

val va = new VectorAssembler().setInputCols(stringColumns\_index).setOutputCol("features")

val labelIndexer = new StringIndexer()

.setInputCol("label")

.setOutputCol("indexedLabel").fit(df)

// Split the data into training and test sets (30% held out for testing)

val Array(trainingData, testData) = df.randomSplit(Array(0.8, 0.2))

val dt = new DecisionTreeClassifier()

.setLabelCol("indexedLabel")

.setFeaturesCol("features")

.setImpurity("entropy")

.setMaxBins(1000)

.setMaxDepth(15)

val labelConverter = new IndexToString()

.setInputCol("prediction")

.setOutputCol("predictedLabel")

.setLabels(labelIndexer.labels)

val stages = index\_transformers :+ va :+ labelIndexer :+ dt :+ labelConverter

val pipeline = new Pipeline()

.setStages(stages)

/\* val pipeline = new Pipeline()

.setStages(Array(indexer, va, dt))\*/

// Train model. This also runs the indexers.

val model = pipeline.fit(trainingData)

val predictions = model.transform(testData)

//predictions.select("predictedLabel", "label", "indexedFeatures").show(5)

predictions.select("predictedLabel", "label", "features").show(5)

}

}