*>>> Please Use Spark Version 2.4.4 to run this code<<<*

*I also Publish our project on data bricks Here’s a link*

[*https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/1213809604916484/1630732396286739/4341593236137157/latest.html*](https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/1213809604916484/1630732396286739/4341593236137157/latest.html)

[*https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/1213809604916484/989478077446330/4341593236137157/latest.html*](https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/1213809604916484/989478077446330/4341593236137157/latest.html)

**Data Exploration and Transformation**

**1.Read dataset in Spark**

val trainDF = spark.read.format("csv").option("inferSchema",true).option("header",true).load("/user/edureka\_836462/bicycleproject/train.csv")

**2.Get summary of data and variable types**

trainDF.printSchema

trainDF.describe().show()

**3.Decide which columns should be categorical and then convert them accordingly**

//Checking Unique values in each column

val exprs = trainDF.schema.fields.filter(x => x.dataType != StringType).map(x=>x.name >"approx\_count\_distinct").toMap

trainDF.agg(exprs).show()

/\* So we are considering "workingday, holiday, season, and wether column" as a categorical column and we are applying onehotencoder on column with values > 2 \*/

val indexer = Array("season","weather").map(c=>new OneHotEncoder().setInputCol(c).setOutputCol(c + "\_Vec"))

val pipeline = new Pipeline().setStages(indexer)

val df\_r = pipeline.fit(trainDF).transform(trainDF).drop("season","weather")

**4.Check for any missing value in data set and treat it**

//There are no missing values

trainDF.select(trainDF.columns.map(c => sum(col(c).isNull.cast("int")).alias(c)): \_\*).show

**5.Explode season column into separate columns such as season\_<val>and drop season**

**6.Execute the same for weather as weather\_<val> and drop weather**

*//we don’t need to explode season column and weather column because previously I already handle categorical column with values > 2 by applying onehotencoder.*

**7. Split datetime in to meaning columns such as hour, day, month, year, etc.**

//Converting datetime string column to timestamp column

val df\_time = df\_r.withColumn("datetime", to\_timestamp(col("datetime"),"d-M-y H:m"))

//Now Spliting date time into meaning columns such as year,month,day,hour

val datetime\_trainDF = df\_time.

withColumn("year", year(col("datetime"))).

withColumn("month", month(col("datetime"))).

withColumn("day", dayofmonth(col("datetime"))).

withColumn("hour", hour(col("datetime"))).

withColumn("minute",minute(col("datetime")))

**8.Explore how count varies with different features such as hour,month,etc**

datetime\_trainDF.groupBy("year").count.show()

datetime\_trainDF.groupBy("month").count.show()

datetime\_trainDF.groupBy("day").count.show()

datetime\_trainDF.groupBy("hour").count.show()

datetime\_trainDF.groupBy("minute").count.show()

**Model Development**

**1.Split the dataset into train and train\_test.**

val splitSeed = 123

val Array(train,train\_test) = datetime\_trainDF.randomSplit(Array(0.7,0.3),splitSeed)

**2. Try different regression algorithms such as linear regression, random forest, etc. and note accuracy.**

//Generate Feature Column

val feature = Array("holiday","workingday","temp","atemp","humidity","windspeed","season\_Vec","weather\_Vec","year","month","day","hour","minute")

//Assemble Feature Column

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

**Linear Regression Model**

//Model Building

val lr = new LinearRegression().setLabelCol("count").setFeaturesCol("features")

//Creating Pipeline

val pipeline = new Pipeline().setStages(Array(assembler,lr))

//Training Model

val lrModel = pipeline.fit(train)

val predictions = lrModel.transform(train\_test)

//Model Summary

val evaluator = new RegressionEvaluator().setLabelCol("count").setPredictionCol("prediction").setMetricName("rmse")

val rmse = evaluator.evaluate(predictions)

println("Linear Regression Root Mean Squared Error (RMSE) on train\_test data = " + rmse)

**Decision Tree Regressor Model**

//Model Building

val dt = new DecisionTreeRegressor().setLabelCol("count").setFeaturesCol("features")

//Creating Pipeline

val pipeline = new Pipeline().setStages(Array(assembler,dt))

//Training Model

val dtModel = pipeline.fit(train)

val predictions = dtModel.transform(train\_test)

//Model Summary

val evaluator = new RegressionEvaluator().setLabelCol("count").setPredictionCol("prediction").setMetricName("rmse")

val rmse = evaluator.evaluate(predictions)

println("Decision Tree Regressor Root Mean Squared Error (RMSE) on train\_test data = " + rmse)

**Random Forest Regressor Model**

//Model Building

val rf = new RandomForestRegressor().setLabelCol("count").setFeaturesCol("features")

//Creating Pipeline

val pipeline = new Pipeline().setStages(Array(assembler,rf))

//Training Model

val rfModel = pipeline.fit(train)

val predictions = rfModel.transform(train\_test)

//Model Summary

val evaluator = new RegressionEvaluator().setLabelCol("count").setPredictionCol("prediction").setMetricName("rmse")

val rmse = evaluator.evaluate(predictions)

println("Random Forest Regressor Root Mean Squared Error (RMSE) on train\_test data = " + rmse)

**GBT Regressor Model**

//Model Building

val gbt = new GBTRegressor().setLabelCol("count").setFeaturesCol("features")

//Creating pipeline

val pipeline = new Pipeline().setStages(Array(assembler,gbt))

//Training Model

val gbtModel = pipeline.fit(train)

val predictions = gbtModel.transform(train\_test)

//Model Summary

val evaluator = new RegressionEvaluator().setLabelCol("count").setPredictionCol("prediction").setMetricName("rmse")

val rmse = evaluator.evaluate(predictions)

println("GBT Regressor Root Mean Squared Error (RMSE) on train\_test data = " + rmse)

**3.Select the best model and persist it.**

//So as we try diferent Regression Alorithms and found that "GBT Regressor Model" is giving better accuracy compare to other.

gbtModel.write.overwrite().save("/user/edureka\_836462/bicycle-model")

**Model Implementation and Prediction**

**Application Development for Model Generation**

For the above steps write an application to:

**1. Clean and Transform the data**

**2. Develop the model and persist it.**

import org.apache.spark.{SparkConf, SparkContext}

import org.apache.spark.SparkContext.\_

import org.apache.spark.sql.\_

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

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

import org.apache.spark.ml.regression.{GBTRegressionModel, GBTRegressor}

import org.apache.spark.ml.evaluation.RegressionEvaluator

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

import org.apache.spark.ml.\_

import org.apache.spark.ml.Pipeline

import org.apache.spark.ml.feature.OneHotEncoder

object BicyclePredict{

def main(args: Array[String]) {

val sparkConf = new SparkConf().setAppName("kapil")

val sc = new SparkContext(sparkConf)

sc.setLogLevel("ERROR")

val spark = new org.apache.spark.sql.SQLContext(sc)

import spark.implicits.\_

println("Reading training data..........................")

val trainDF = spark.read.format("csv").

option("inferSchema",true).

option("header",true).

load("/user/edureka\_836462/bicycleproject/train.csv ")

println("Cleaning data.........................")

//Converting datetime string column to timestamp column

val df\_time = trainDF.withColumn("datetime", to\_timestamp(col("datetime"),"d-M-y H:m"))

//Now Spliting date time into meaning columns such as year,month,day,hour

val datetime\_trainDF = df\_time.

withColumn("year", year(col("datetime"))).

withColumn("month", month(col("datetime"))).

withColumn("day", dayofmonth(col("datetime"))).

withColumn("hour", hour(col("datetime"))).

withColumn("minute",minute(col("datetime")))

//Onehot encoding on season and weather column.

val indexer = Array("season","weather").map(c=>new OneHotEncoder().setInputCol(c).setOutputCol(c + "\_Vec"))

val pipeline = new Pipeline().setStages(indexer)

val df\_r = pipeline.fit(datetime\_trainDF).transform(datetime\_trainDF)

//split data into train test

val splitSeed =123

val Array(train, train\_test) = df\_r.randomSplit(Array(0.7, 0.3), splitSeed)

//Generate Feature Column

val feature\_cols = Array("holiday","workingday","temp","atemp","humidity","windspeed","season\_Vec","weather\_Vec","year","month","day","hour","minute")

//Assemble Feature

val assembler = new VectorAssembler().setInputCols(feature\_cols).setOutputCol("features")

//Model Building

val gbt = new GBTRegressor().setLabelCol("count").setFeaturesCol("features")

val pipeline2 = new Pipeline().setStages(Array(assembler,gbt))

println("Training model.........................")

val gbt\_model = pipeline2.fit(train)

val predictions = gbt\_model.transform(train\_test)

val evaluator = new RegressionEvaluator().setLabelCol("count").setPredictionCol("prediction").setMetricName("rmse")

val rmse = evaluator.evaluate(predictions)

println("GBT Regressor Root Mean Squared Error (RMSE) on train\_test data = " + rmse)

println("Persisting the model................")

gbt\_model.write.overwrite().save("/user/edureka\_836462/bicycle-model ")

}

}

**Application Execution**

spark2-submit --class "BicyclePredict" --master yarn /mnt/home/edureka\_836462/BicycleProject/BicycleTrain/target/scala-2.11/bicycletrain\_2.11-1.0.jar

**Application Development for Demand Prediction**

Model Prediction Application – Write an application to predict the bike demand based on the input dataset from HDFS:

**1. Load the persisted model.**

**2. Predict bike demand**

**3. Persist the result to RDBMS**

**Database creation:**

mysql -h mysqldb.edu.cloudlab.com -u labuser -p edureka

create database kapil64\_bicycle;

CREATE TABLE predictions (

datetime datetime,

count FLOAT

);

**Prediction Application:**

import org.apache.spark.{SparkConf, SparkContext}

import org.apache.spark.SparkContext.\_

import org.apache.spark.sql.\_

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

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

import org.apache.spark.ml.regression.{GBTRegressionModel, GBTRegressor}

import org.apache.spark.ml.evaluation.RegressionEvaluator

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

import org.apache.spark.ml.\_

import org.apache.spark.ml.Pipeline

import org.apache.spark.ml.feature.OneHotEncoder

object BicyclePredict {

def main(args: Array[String]) {

val sparkConf = new SparkConf().setAppName("Telecom")

val sc = new SparkContext(sparkConf)

sc.setLogLevel("ERROR")

val spark = new org.apache.spark.sql.SQLContext(sc)

import spark.implicits.\_

println("Reading Training data.................")

val testDF = spark.read.format("csv").

option("inferSchema",true).

option("header",true).

load("/user/edureka\_836462/bicycleproject/test.csv ")

println("Cleaning data.................")

//Converting datetime string column to timestamp column

val df\_time = testDF.withColumn("datetime", to\_timestamp(col("datetime"),"d-M-y H:m"))

//Now Spliting date time into meaning columns such as year,month,day,hour

val datetime\_testDF = df\_time.

withColumn("year", year(col("datetime"))).

withColumn("month", month(col("datetime"))).

withColumn("day", dayofmonth(col("datetime"))).

withColumn("hour", hour(col("datetime"))).

withColumn("minute",minute(col("datetime")))

//Onehot encoding on season and weather column.

val indexer = Array("season","weather").map(c=>new OneHotEncoder().setInputCol(c).setOutputCol(c + "\_Vec"))

val pipeline = new Pipeline().setStages(indexer)

val df\_r = pipeline.fit(datetime\_testDF).transform(datetime\_testDF)

println("Loading Trained Model..................")

val gbt\_model = PipelineModel.read.load("/user/edureka\_836462/bicycle-model ")

println("Making predictions....................")

val predictions = gbt\_model.transform(df\_r).select($"datetime",$"prediction".as("count"))

println("Persisting the result to RDBMS.................")

predictions.write.format("jdbc").

option("url", "jdbc:mysql://mysqldb.edu.cloudlab.com/kapil64\_bicycle").

option("driver", "com.mysql.cj.jdbc.Driver").option("dbtable", "predictions").

option("user", "labuser").

option("password", "edureka").

mode(SaveMode.Append).save

}

}

**Run the application:**

spark2-submit --packages mysql:mysql-connector-java:8.0.13 --class "BicyclePredict" --master yarn /mnt/home/edureka\_836462/BicycleProject/BicyclePredict/target/scala-2.11/bicyclepredict\_2.11-1.0.jar

**Application for Streaming Data**

Write an application to predict demand on streaming data:

**1. Setup flume to push data into spark flume sink.**

**Kafka topic creation:**

kafka-topics --create --zookeeper ip-20-0-21-161.ec2.internal:2181 --replication-factor 1 --partitions 1 --topic edureka\_836462\_bicycle\_kapil

**Flume configuration:**

agent1.sources = source1

agent1.channels = channel1

agent1.sinks = spark

agent1.sources.source1.type = org.apache.flume.source.kafka.KafkaSource

agent1.sources.source1.kafka.bootstrap.servers = ip-20-0-31-210.ec2.internal:9092

agent1.sources.source1.kafka.topics = edureka\_836462\_bicycle\_kapil

agent1.sources.source1.kafka.consumer.group.id = edureka\_836462\_bicycle\_kapil

agent1.sources.source1.channels = channel1

agent1.sources.source1.interceptors = i1

agent1.sources.source1.interceptors.i1.type = timestamp

agent1.sources.source1.kafka.consumer.timeout.ms = 100

agent1.channels.channel1.type = memory

agent1.channels.channel1.capacity = 10000

agent1.channels.channel1.transactionCapacity = 1000

agent1.sinks.spark.type = org.apache.spark.streaming.flume.sink.SparkSink

agent1.sinks.spark.hostname = ip-20-0-41-62.ec2.internal

agent1.sinks.spark.port = 4143

agent1.sinks.spark.channel = channel1

**Run Flume agent:**

flume-ng agent --conf conf --conf-file bicycle.conf --name agent1 -Dflume.root.logger=DEBUG,console

**2. Configure spark streaming to pulldata from spark flume sink using receivers and predict the demand using model and persist the result to RDBMS.**

import org.apache.spark.{SparkConf, SparkContext}

import org.apache.spark.SparkContext.\_

import org.apache.spark.sql.\_

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

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

import org.apache.spark.ml.regression.{GBTRegressionModel, GBTRegressor}

import org.apache.spark.ml.feature.{StringIndexer, VectorAssembler}

import org.apache.spark.ml.\_

import org.apache.spark.streaming.{Seconds, StreamingContext}

import org.apache.spark.streaming.flume.\_

import org.apache.spark.ml.Pipeline

import org.apache.spark.ml.feature.OneHotEncoder

object BicycleStreaming {

case class Bicycle(datetime: String, season: Int, holiday: Int, workingday: Int, weather: Int, temp: Double, atemp: Double, humidity: Int, windspeed: Double)

def main(args: Array[String]) {

val sparkConf = new SparkConf().setAppName("kapil")

val sc = new SparkContext(sparkConf)

val ssc = new StreamingContext(sc, Seconds(2))

sc.setLogLevel("ERROR")

val spark = new org.apache.spark.sql.SQLContext(sc)

import spark.implicits.\_

val flumeStream = FlumeUtils.createPollingStream(ssc, "ip-20-0-41-62.ec2.internal", 4143)

println("Loading tained model.............")

val gbt\_model = PipelineModel.read.load("/user/edureka\_836462/bicycle-model")

val lines = flumeStream.map(event => new String(event.event.getBody().array(), "UTF-8"))

lines.foreachRDD { rdd =>

def row(line: List[String]): Bicycle = Bicycle(line(0), line(1).toInt, line(2).toInt,

line(3).toInt, line(4).toInt, line(5).toDouble, line(6).toDouble, line(7).toInt,

line(8).toDouble

)

val rows\_rdd = rdd.map(\_.split(",").to[List]).map(row)

val rows\_df = rows\_rdd.toDF

if(rows\_df.count > 0) {

val df\_time = rows\_df.withColumn("datetime",to\_timestamp(col("datetime"),"d-M-y H:m"))

val datetime\_testDF = df\_time.

withColumn("year", year(col("datetime"))).

withColumn("month", month(col("datetime"))).

withColumn("day", dayofmonth(col("datetime"))).

withColumn("hour", hour(col("datetime"))).

withColumn("minute",minute(col("datetime")))

//Onehot encoding on season nd weather column

val indexer = Array("season","weather").map(c => new OneHotEncoder().setInputCol(c).setOutputCol(c + "\_Vec"))

val pipeline = new Pipeline().setStages(indexer)

val df\_r = pipeline.fit(datetime\_testDF).transform(datetime\_testDF)

println("Making predictions...............")

val predictions = gbt\_model.transform(df\_r).select($"datetime",$"prediction".as("count"))

println("Persisting the result to RDBMS..................")

predictions.write.format("jdbc").

option("url", "jdbc:mysql://mysqldb.edu.cloudlab.com/kapil64\_bicycle").

option("driver", "com.mysql.cj.jdbc.Driver").option("dbtable", "predictions").

option("user", "labuser").

option("password", "edureka").

mode(SaveMode.Append).save

}

}

ssc.start()

ssc.awaitTermination()

}

}

**Run the application:**

spark2-submit --packages mysql:mysql-connector-java:8.0.13 --class "BicycleStreaming" --master yarn /mnt/home/edureka\_836462/BicycleProject/BicycleStreaming/target/scala-2.11/bicyclestreaming\_2.11-1.0.jar

**3. Push messages from flume to test the application. Here application should process and persist the result to RDBMS.**

kafka-console-producer --broker-list ip-20-0-31-210.ec2.internal:9092 --topic edureka\_836462\_bicycle\_kapil

**//Pushing Message**

1/20/2011 0:00,1,0,1,1,10.66,11.365,56,26.0027