8. Spark ML

목 차

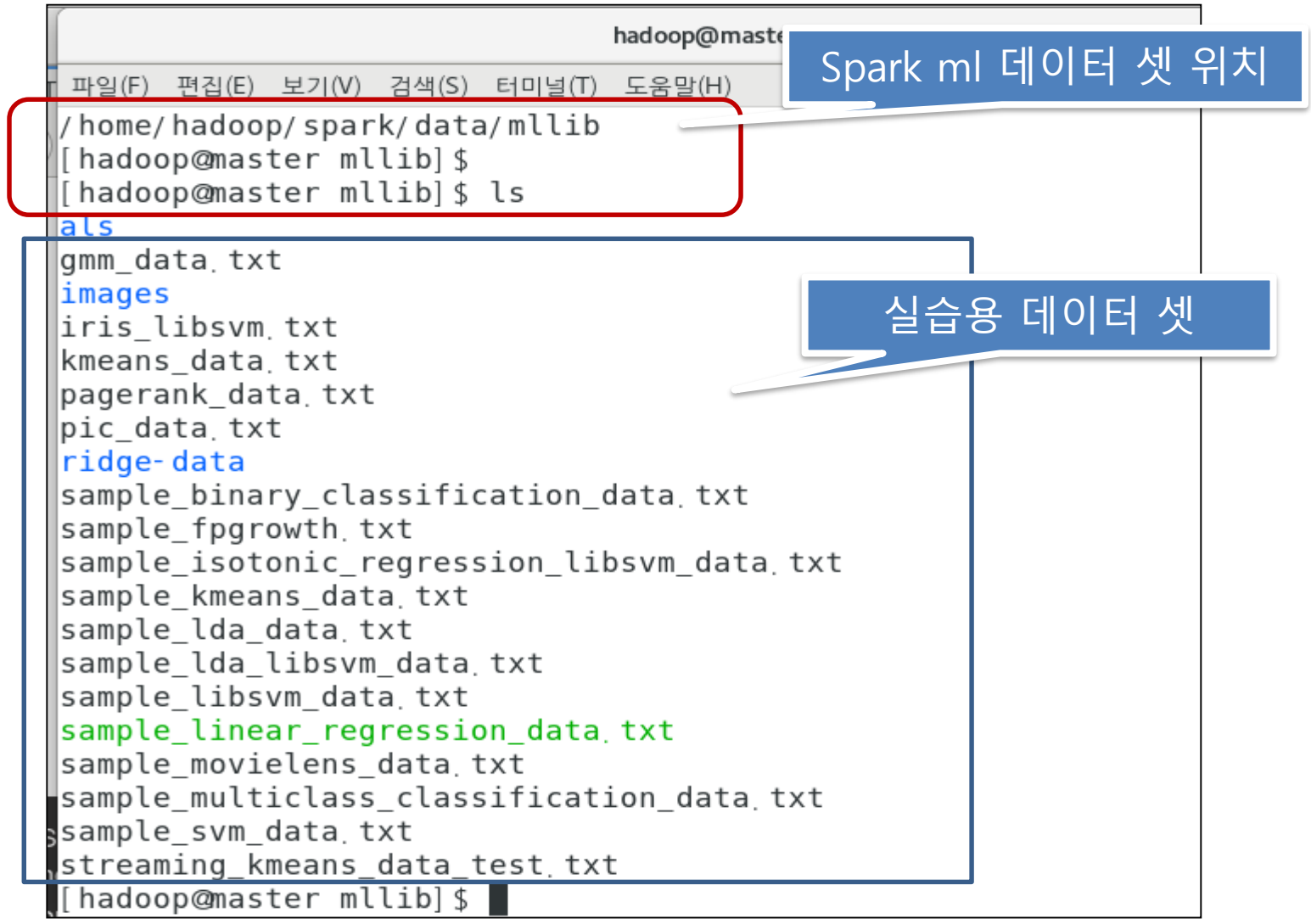
1. Spark ML 개요

- Spark2.0 ML 라이브러리
  - ✓ 분류, 회귀, 클러스터링, 협업 필터링과 같은 일반적인 머신러닝 알고리즘(심층 신경망(Deep Neural Network) 없음)과 함께 특징 추출, 변형, 차원 감소 및 선택을 위한 도구, ML 파이프라인 구축과 평가, 튜닝을 위한 도구를 제공
  - ✓ 알고리즘과 모델 및 파이프라인의 저장/로드, 데이터 처리, 선형 대수학과 통계학 수행을 위한 유틸리티도 포함
  - ✓ 스파크 ML은 스칼라(Scala)로 작성됐으며 선형 대수학 패키지인 브리즈(Breeze)를 사용

2. Spark ML 환경

- Spark ML 작업 환경
  - ✓ 실습용 데이터 셋 경로
  - ✓ Spark API(jar) 경로

Spark ML 데이터 셋



[hadoop@master mllib]$ cat iris_libsvm.txt

```
0.0 1:5.1 2:3.5 3:1.4 4:0.2     -> 칼럼index.value 칼럼index:value
0.0 1:4.9 2:3.0 3:1.4 4:0.2
0.0 1:4.7 2:3.2 3:1.3 4:0.2
0.0 1:4.6 2:3.1 3:1.5 4:0.2
0.0 1:5.0 2:3.6 3:1.4 4:0.2
0.0 1:5.4 2:3.9 3:1.7 4:0.4
0.0 1:4.6 2:3.4 3:1.4 4:0.3
0.0 1:5.0 2:3.4 3:1.5 4:0.2
0.0 1:4.4 2:2.9 3:1.4 4:0.2
0.0 1:4.9 2:3.1 3:1.5 4:0.1
:
2.0 1:6.8 2:3.2 3:5.9 4:2.3
2.0 1:6.7 2:3.3 3:5.7 4:2.5
2.0 1:6.7 2:3.0 3:5.2 4:2.3
2.0 1:6.3 2:2.5 3:5.0 4:1.9
2.0 1:6.5 2:3.0 3:5.2 4:2.0
2.0 1:6.2 2:3.4 3:5.4 4:2.3
2.0 1:5.9 2:3.0 3:5.1 4:1.8
```

[hadoop@master mllib]$ cat kmeans_data.txt

```
0.0 0.0 0.0
0.1 0.1 0.1
0.2 0.2 0.2
9.0 9.0 9.0
9.1 9.1 9.1
9.2 9.2 9.2
```

[hadoop@master mllib]$ cat sample_linear_regression_data.txt

```
0.4250502150408626 1:0.7771717566171905 2:-0.8729202752916785 3:-0.25782888805127024 4:-
0.13605474993771205 5:0.5911781118120025 6:-0.8444023967853633 7:0.6787302541469229 8:-
0.5444299313083194 9:0.356121883138657 10:-0.8845333845080687
-0.8743487925900991 1:-0.9087681208947878 2:-0.292625136739453 3:-0.35113758823291774 4:-
0.705933223571676 5:-0.6882289471031144 6:0.8350131255297044 7:-0.7659016065609232
8:0.11400114955653207 9:-0.9466143658505732 10:-0.5033643125229932
-5.615143641864686 1:-0.6688289820084299 2:-0.4623159855015393 3:0.012827807007503855 4:-
0.44521264878006117 5:-0.5563111031201406 6:-0.6065295981983794 7:0.3806712426786838 8:-
0.11317152118817408 9:0.507896127467435 10:-0.8487801189674464
-0.1829397047693725 1:0.09377558075225512 2:0.5774384503027374 3:-0.7104684187448009 4:-
0.07285914169135976 5:-0.8797920488335114 6:0.6099615504974201 7:-0.8047440624324915 8:-
0.6877856114263066 9:0.5843004021777447 10:0.5190581455348131
18.479680552020344 1:0.9635517137863321 2:0.9954507816218203 3:0.11959899129360774
4:0.3753283274192787 5:-0.9386713095183621 6:0.0926833703812433 7:0.48003949462701323
8:0.9432769781973132 9:-0.9637036991931129 10:-0.4064407447273508
1.3850645873427236 1:0.14476184437006356 2:-0.11280617018445871 3:-0.4385084538142101 4:-
0.5961619435136434 5:0.419554626795412 6:-0.5047767472761191 7:0.457180284958592 8:-
0.9129360314541999 9:-0.6320022059786656 10:-0.44989608519659363
```

Spark ML library(jar 파일)

```
[hadoop@master ~]$
[hadoop@master ~]$ cd spark
[hadoop@master spark]$
[hadoop@master spark]$ ls
LICENSE  R        RELEASE  conf  examples  kubernetes  python  yarn
NOTICE   README.md  bin      data  jars      licenses    sbin
[hadoop@master spark]$
[hadoop@master spark]$ cd jars/
[hadoop@master jars]$
[hadoop@master jars]$ pwd
/home/hadoop/spark/jars
[hadoop@master jars]$ ls
JavaEWAH-0.3.2.jar                    javax.ws.rs-api-2.0.1.jar
RoaringBitmap-0.5.11.jar              javolution-5.5.1.jar
ST4-4.0.4.jar                         jaxb-api-2.2.2.jar
activation-1.1.1.jar                  jcl-over-slf4j-1.7.16.jar
aircompressor-0.8.jar                 jdo-api-3.0.1.jar
antlr-2.7.7.jar                       jersey-client-2.22.2.jar
antlr-runtime-3.4.jar                 jersey-common-2.22.2.jar
antlr4-runtime-4.7.jar                jersey-container-servlet-2.22.2.jar
aopalliance-1.0.jar                   jersey-container-servlet-core-2.22.2.jar
aopalliance-repackaged-2.4.0-b34.jar  jersey-guava-2.22.2.jar
```

Spark 관련 라이브러리

3. Spark ML Sample

https://spark.apache.org/docs/2.2.0/ml-classification-regression.html

LinearRegression

```
import org.apache.spark.ml.regression.LinearRegression

// Load training data
val training = spark.read.format("libsvm")
  .load("data/mllib/sample_linear_regression_data.txt")

val lr = new LinearRegression() // 선형회귀모델 생성
  .setMaxIter(10)  // hyper parameters
  .setRegParam(0.3)
  .setElasticNetParam(0.8)

// Fit the model
val lrModel = lr.fit(training)

// Print the coefficients and intercept for linear regression – 기울기와 절편
println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")

// Summarize the model over the training set and print out some metrics
val trainingSummary = lrModel.summary
println(s"numIterations: ${trainingSummary.totalIterations}")
println(s"objectiveHistory: [${trainingSummary.objectiveHistory.mkString(",")}]")

// model 평가
trainingSummary.residuals.show() // 잔차
println(s"RMSE: ${trainingSummary.rootMeanSquaredError}")
println(s"r2: ${trainingSummary.r2}")
```

LogisticRegression

```
import org.apache.spark.ml.classification.LogisticRegression

// Load training data
val training = spark.read.format("libsvm").load("data/mllib/sample_libsvm_data.txt")

val lr = new LogisticRegression()
  .setMaxIter(10)
  .setRegParam(0.3)
  .setElasticNetParam(0.8)

// Fit the model
val lrModel = lr.fit(training)

// Print the coefficients and intercept for logistic regression
println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")

// We can also use the multinomial family for binary classification
val mlr = new LogisticRegression()
  .setMaxIter(10)
  .setRegParam(0.3)
  .setElasticNetParam(0.8)
  .setFamily("multinomial")

val mlrModel = mlr.fit(training)

// Print the coefficients and intercepts for logistic regression with multinomial family
println(s"Multinomial coefficients: ${mlrModel.coefficientMatrix}")
println(s"Multinomial intercepts: ${mlrModel.interceptVector}")
```

DecisionTreeClassification

```
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.classification.DecisionTreeClassificationModel
import org.apache.spark.ml.classification.DecisionTreeClassifier
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.spark.ml.feature.{IndexToString, StringIndexer, VectorIndexer}

// Load the data stored in LIBSVM format as a DataFrame.
val data = spark.read.format("libsvm").load("data/mllib/sample_libsvm_data.txt")

// Index labels, adding metadata to the label column.
// Fit on whole dataset to include all labels in index.
val labelIndexer = new StringIndexer()
  .setInputCol("label")
  .setOutputCol("indexedLabel")
  .fit(data)
// Automatically identify categorical features, and index them.
val featureIndexer = new VectorIndexer()
  .setInputCol("features")
  .setOutputCol("indexedFeatures")
  .setMaxCategories(4) // features with > 4 distinct values are treated as continuous.
  .fit(data)

// Split the data into training and test sets (30% held out for testing).
val Array(trainingData, testData) = data.randomSplit(Array(0.7, 0.3))

// Train a DecisionTree model.
val dt = new DecisionTreeClassifier()
  .setLabelCol("indexedLabel")
  .setFeaturesCol("indexedFeatures“)
```

```
// Convert indexed labels back to original labels.
val labelConverter = new IndexToString()
  .setInputCol("prediction")
  .setOutputCol("predictedLabel“)
  .setLabels(labelIndexer.labels)

// Chain indexers and tree in a Pipeline.
val pipeline = new Pipeline().setStages(Array(labelIndexer, featureIndexer, dt, labelConverter))

// Train model. This also runs the indexers.
val model = pipeline.fit(trainingData)

// Make predictions.
val predictions = model.transform(testData)

// Select example rows to display.
predictions.select("predictedLabel", "label", "features").show(5)

// Select (prediction, true label) and compute test error.
val evaluator = new MulticlassClassificationEvaluator()
  .setLabelCol("indexedLabel“).setPredictionCol("prediction“).setMetricName("accuracy")
val accuracy = evaluator.evaluate(predictions)
println("Test Error = " + (1.0 - accuracy))

val treeModel = model.stages(2).asInstanceOf[DecisionTreeClassificationModel]
println("Learned classification tree model:₩n" + treeModel.toDebugString)
```

RandomForestClassification

```
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.classification.{RandomForestClassificationModel, RandomForestClassifier}
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.spark.ml.feature.{IndexToString, StringIndexer, VectorIndexer}

// Load and parse the data file, converting it to a DataFrame.
val data = spark.read.format("libsvm").load("data/mllib/sample_libsvm_data.txt")

// Index labels, adding metadata to the label column.
// Fit on whole dataset to include all labels in index.
val labelIndexer = new StringIndexer()
  .setInputCol("label")
  .setOutputCol("indexedLabel")
  .fit(data)
// Automatically identify categorical features, and index them.
// Set maxCategories so features with > 4 distinct values are treated as continuous.
val featureIndexer = new VectorIndexer()
  .setInputCol("features")
  .setOutputCol("indexedFeatures")
  .setMaxCategories(4)
  .fit(data)

// Split the data into training and test sets (30% held out for testing).
val Array(trainingData, testData) = data.randomSplit(Array(0.7, 0.3))

// Train a RandomForest model.
val rf = new RandomForestClassifier()
  .setLabelCol("indexedLabel")
  .setFeaturesCol("indexedFeatures")
  .setNumTrees(10)
```

```
// Convert indexed labels back to original labels.
val labelConverter = new IndexToString()
  .setInputCol("prediction")
  .setOutputCol("predictedLabel")
  .setLabels(labelIndexer.labels)

// Chain indexers and forest in a Pipeline.
val pipeline = new Pipeline()
  .setStages(Array(labelIndexer, featureIndexer, rf, labelConverter))

// Train model. This also runs the indexers.
val model = pipeline.fit(trainingData)

// Make predictions.
val predictions = model.transform(testData)

// Select example rows to display.
predictions.select("predictedLabel", "label", "features").show(5)

// Select (prediction, true label) and compute test error.
val evaluator = new MulticlassClassificationEvaluator()
  .setLabelCol("indexedLabel")
  .setPredictionCol("prediction")
  .setMetricName("accuracy")
val accuracy = evaluator.evaluate(predictions)
println("Test Error = " + (1.0 - accuracy))

val rfModel = model.stages(2).asInstanceOf[RandomForestClassificationModel]
println("Learned classification forest model:₩n" + rfModel.toDebugString)
```

KMeans model

```
import org.apache.spark.mllib.clustering.KMeans  // Kmeans model 생성
import org.apache.spark.mllib.linalg.Vectors // Vector 생성
val sparkHome = sys.env("SPARK_HOME“)
val data = sc.textFile("file://"+sparkHome + "/data/mllib/kmeans_data.txt") // local file read
val parseData = data.map(s => Vectors.dense(s.split(' ').map(_.toDouble))).cache()

for(line <- parseData) println(line)
parseData.foreach(line => println(line))

val numClusters = 2
val numIterations = 20
val kmeans_model = KMeans.train(parseData, numClusters, numIterations)

kmeans_model.k
kmeans_model.clusterCenters

// test data 생성
val test_data1 = Vectors.dense(0.3,0.3,0.3)
val test_data2 = Vectors.dense(8.0,8.0,8.0)

// model test
kmeans_model.predict(test_data1)
kmeans_model.predict(test_data2)

for(line <- parseData) println(line + "=>" + kmeans_model.predict(line))
parseData.foreach(line => println(line + "=>" + kmeans_model.predict(line))
val kmeans_pred = parseData.map(line => kmeans_model.predict(line))
// hdfs save
kmeans_pred.saveAsTextFile("hdfs://master:9000/output/kmeans“)
```