Python 数据分析与数据挖掘（Python for Data Analysis&Data Mining）

Chap 17 - Spark 大数据挖掘

- Q: 如何解决大数据下的挖掘实现？
- A：Spark MLlib

内容：

- Spark框架概述
- Spark MLlib 分类模型的应用和性能比较
- Spark MLlib 回归模型的应用
- Spark MLlib 聚类模型的应用
- Spark MLlib 关联规则挖掘
- 应用领域：本学期课程中的分类、回归、聚类算法解决的问题都适用

实践：

- 分类算法：支持向量机（Support Vector Machine，SVM），决策树分类（Decision Tree, DT），朴素贝叶斯（Naive Bayes，NB）算法，随机森林（Random Forest），LogReg；
- 回归算法：线性回归，广义线性回归，决策树回归等回归算法；
- 聚类算法：K-Means算法，LDA主题模型
- 关联规则算法：FP-增长算法

这节课解决大数据背景下数据挖掘的实践问题。本节课采用Spark MLlib框架，通过使用 MLlib API 实现本学期涵盖的多个分类、回归、聚类算法和频繁模式挖掘算法，适用于本学期中的多种数据类型。本节课的实例需要预先安装Hadoop和Spark（内含MLlib机器学习库）。

1. 概述

下载+安装

Spark 2.1.1 的官方下载地址 (http://spark.apache.org/downloads.html)（目前现在的版本是 spark-2.1.1-bin-hadoop2.7）

Hadoop 2.7 的官方下载地址 (http://hadoop.apache.org/releases.html)（与Spark下载的版本对应）

下载安装Spark后，MLlib是其中的一个模块，专门进行可扩展的机器学习。

为什么使用MLlib？

MLlib 和ML是构建在Apache Spark之上，一个专门针对大量数据处理的通用的、快速的引擎，是Spark的可扩展机器学习库。

1. 使用方便
    - Java, Scala, Python, 和 R。从 Spark 0.9 版本，MLlib 支持 Python 的 NumPy； 从 Spark 1.5开始，支持 R；
    - 可以使用任何 Hadoop 数据源 (如，HDFS, HBase, 或本地文件), 很容易嵌入到 Hadoop workflows
2. 性能高
    - 算法性能高，比 MapReduce 快100倍
3. 部署容易
    - 可以运行在现有的Hadoop集群和数据； 如果已经安装 Hadoop 2 cluster,则不需要预安装，直接可以运行 Spark 和 MLlib

MLlib 包括通用的学习算法和工具类

包括分类，回归，聚类，协同过滤，降维，调优等

2. 大数据挖掘库

MLlib库概述

分类算法的实现：

- SVM (Support Vector Machine)
- LogReg (Logistic Regression)
- DT (Decision Tree)
- NB (Naive Bayes)
- Random Forest
- Multilayer perceptron classifier

回归算法的实现:

- Linear regression
- Generalized Linear Regression
- DT regression （Decision Tree Regression）

聚类算法的实现

- K-Means
- LDA主题模型

关联规则挖掘

- FPGrowth算法

注意：

- 旧版 Spark MLlib 是个 RDD-based API (是 spark.mllib 库)
- 目前的Spark MLlib 是 DataFrame-based API (即 spark.ml 库), 这是当前MLlib最主要的API
- 从 import 的库可以区分

2.1 Spark MLlib中的分类算法

1）Spark MLlib中的SVM （只有RDD-based）

SPARK_HOME=/home/lanman/Hadoop/hadoop-2.7.3/spark-2.1.1-bin-hadoop2.7

将下面的代码存入svmSparkMLlib.py中，然后提交下面的spark任务：

- cd $SPARK_HOME
- ./bin/spark-submit --master local[8] /home/lanman/course/dm/svmSparkMLlib.py
- 在本地机运行8个线程 Run on local machine using 8 threads

In [ ]:

```
# -*- coding: utf-8 -*-

from pyspark import SparkContext
from pyspark.sql import SQLContext
from pyspark.mllib.classification import SVMWithSGD, SVMModel
from pyspark.mllib.regression import LabeledPoint

# Load and parse the data
def parsePoint(line):
    values = [float(x) for x in line.split('\t')]  # 特征以tab分隔，最后一列是类标
    return LabeledPoint(int(values[-1]), values[:-1])  # 第一个是类标class label，第二个是featur
es

# 创建sc环境
sc = SparkContext()
sqlContext = SQLContext(sc)

# 解析training data
trainingData = sc.textFile("/home/lanman/course/dm/data/horseColicTraining.txt")
# 使用绝对路径，或者$SPARK_HOME目录下路径
trainingParsedData = trainingData.map(parsePoint)

# Build the model on Training data
model = SVMWithSGD.train(trainingParsedData, iterations=100)

# 解析test data
testData = sc.textFile("/home/lanman/course/dm/data/horseColicTest.txt")
# 使用绝对路径，或者$SPARK_HOME目录下路径
testParsedData = testData.map(parsePoint)

# Evaluating the model on Test data
labelsAndPreds = testParsedData.map(lambda p: (p.label, model.predict(p.features)))
testErr = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(testParsedData.count())

print "Test Error = %2.4f%%" % (float(testErr)*100)

# SVM的结果，默认参数
#Test Error = 50.7463% （iterations=100）took 0.075601 s
#Test Error = 49.2537%（iterations=300）took 0.068854 s

# Save and load model
#model.save(sc, "/home/lanman/course/dm/model/mySVMModelPath")
#sameModel = LogisticRegressionModel.load(sc, "/home/lanman/course/dm/model/mySVMModelPath")
```

2 ） Spark MLlib 中的 LogReg （ RDD-based ）

SPARK_HOME=/home/lanman/Hadoop/hadoop-2.7.3/spark-2.1.1-bin-hadoop2.7

将下面的代码存入logRegSparkMLlib.py中，然后提交下面的spark任务：

- cd $SPARK_HOME
- ./bin/spark-submit --master local[8] /home/lanman/course/dm/logRegSparkMLlib.py
- 在本地机运行8个线程 Run on local machine using 8 threads

In [ ]:

```
# -*- coding: utf-8 -*-

from pyspark import SparkContext
from pyspark.sql import SQLContext
from pyspark.mllib.classification import LogisticRegressionWithLBFGS, LogisticRegressionModel
from pyspark.mllib.regression import LabeledPoint

# Load and parse the data
def parsePoint(line):
    values = [float(x) for x in line.split('\t')]  # 特征以tab分隔，最后一列是类标
    return LabeledPoint(int(values[-1]), values[:-1])  # 第一个是类标class label，第二个是featur
es

# 创建sc环境
sc = SparkContext()
sqlContext = SQLContext(sc)

# 解析training data
trainingData = sc.textFile("/home/lanman/course/dm/data/horseColicTraining.txt")
# 使用绝对路径，或者$SPARK_HOME目录下路径
trainingParsedData = trainingData.map(parsePoint)

# Build the model on Training data
model = LogisticRegressionWithLBFGS.train(trainingParsedData, iterations=300)

# 解析test data
testData = sc.textFile("/home/lanman/course/dm/data/horseColicTest.txt")
# 使用绝对路径，或者$SPARK_HOME目录下路径
testParsedData = testData.map(parsePoint)

# Evaluating the model on Test data
labelsAndPreds = testParsedData.map(lambda p: (p.label, model.predict(p.features)))
testErr = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(testParsedData.count())

print "Test Error = %2.4f%%" % (float(testErr)*100)


# LogReg的结果
#Test Error = 26.8657% took 0.048703 s s (iterations=100)
#Test Error = 26.8657% took 0.055063 s s (iterations=300)

# Save and load model
#model.save(sc, "/home/lanman/course/dm/model/myLogRegSparkModelPath")
#sameModel = LogisticRegressionModel.load(sc, "/home/lanman/course/dm/model/myLogRegSparkModelPa
th")
```

运行 ./bin/spark-submit --master local[8] /home/lanman/course/dm/svmSparkMLlib.py的结果如下：

SVM的结果，默认参数

- Test Error = 50.7463% （iterations=100）运行时间大概0.076秒

修改代码里面的迭代次数（iterations=300）：

- Test Error = 49.2537%（iterations=300）运行时间大概 0.069秒

对比LogReg的结果

运行 ./bin/spark-submit --master local[8] /home/lanman/course/dm/logRegSparkMLlib.py的结果如下：

LogReg的结果

- Test Error = 26.8657%， 运行时间约 0.05 s
- 改变迭代次数对准确率无影响。

3） Spark MLlib 中的 LogReg （DataFrame-based）

SPARK_HOME=/home/lanman/Hadoop/hadoop-2.7.3/spark-2.1.1-bin-hadoop2.7

将下面的代码存入logRegSparkML.py中，然后提交下面的spark任务：

- cd $SPARK_HOME
- ./bin/spark-submit --master local[8] /home/lanman/course/dm/logRegSparkML.py
- 在本地机运行8个线程 Run on local machine using 8 threads

In [ ]:

```
# -*- coding: utf-8 -*-
from pyspark.ml.classification import LogisticRegression

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("LogisticRegressionSparkML").getOrCreate()

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

lr = LogisticRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)

# Fit the model
lrModel = lr.fit(training)

# Print the coefficients and intercept for logistic regression
print "Coefficients: %s" % str(lrModel.coefficients)
print "Intercept: %s" % str(lrModel.intercept)

# We can also use the multinomial family for binary classification
mlr = LogisticRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8, family="multinomial")

# Fit the model
mlrModel = mlr.fit(training)

# Print the coefficients and intercepts for logistic regression with multinomial family
print "Multinomial coefficients: %s" % str(mlrModel.coefficientMatrix)
print "Multinomial intercepts: %s" % str(mlrModel.interceptVector)

spark.stop()
```

运行 Logistic Regression （DataFrame-based）的结果

- Multinomial coefficients:

  ```
  DenseMatrix([[ 0.,   0.,   0., ...,   0.,   0.,   0.],
               [ 0.,   0.,   0., ...,   0.,   0.,   0.]])
  ```

- Multinomial intercepts: [-0.120658794459,0.120658794459]

4）Spark MLlib中的 DT （Decision Tree）算法 (RDD-based)

将下面的代码存入 DTSparkMLlib.py中，然后提交下面的spark任务：

- cd $SPARK_HOME
- ./bin/spark-submit --master local[8] /home/lanman/course/dm/DTSparkMLlib.py
- 在本地机运行8个线程 Run on local machine using 8 threads

In [ ]:

```
# -*- coding: utf-8 -*-
from pyspark import SparkContext
from pyspark.sql import SQLContext

from pyspark.mllib.tree import DecisionTree, DecisionTreeModel
from pyspark.mllib.util import MLUtils

# 创建sc环境
sc = SparkContext()
sqlContext = SQLContext(sc)

# Load and parse the data file into an RDD of LabeledPoint.
data = MLUtils.loadLibSVMFile(sc, 'data/mllib/sample_libsvm_data.txt') # 数据默认存放在 SPARK_HO
ME 目录下

# Split the data into training and test sets (30% held out for testing)
(trainingData, testData) = data.randomSplit([0.7, 0.3])  # 随机分隔数据

# Train a DecisionTree model.
#  Empty categoricalFeaturesInfo indicates all features are continuous.
model = DecisionTree.trainClassifier(trainingData, numClasses=2, categoricalFeaturesInfo={},
                                     impurity='gini', maxDepth=5, maxBins=32)

# Evaluate model on test instances and compute test error
predictions = model.predict(testData.map(lambda x: x.features))
labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)
testErr = (labelsAndPredictions.filter(lambda (v, p): v != p).count() / float(testData.count()))

print "Test Error = %2.4f%%" % (float(testErr)*100)

#print "Learned classification tree model:"
#print model.toDebugString()

# Test Error = 3.85%  took 0.103933 s

# Save and load model
#model.save(sc, "target/tmp/myDecisionTreeClassificationModel")
#sameModel = DecisionTreeModel.load(sc, "target/tmp/myDecisionTreeClassificationModel")
```

注意： 在上面的DT算法的 DTSparkMLlib.py实现代码中，因为数据是被随机分为train和test数据集，因此，每次运行代码得到的结果都不相同。

运行 Decision Tree (RDD-based)的结果

- Test Error = 3.45% took 0.025 s
- Test Error = 0.00% took 0.089 s
- Test Error = 2.94% took 0.100 s
- Test Error = 3.85% took 0.074 s

```
17/06/03 16:45:37 INFO DAGScheduler: Job 8 finished: count at /home/lanman/cour
se/dm/treesSparkMLlib.py:26, took 0.025007 s
Test Error = 3.4483%
Learned classification tree model:
DecisionTreeModel classifier of depth 2 with 5 nodes
  If (feature 434 <= 0.0)
   If (feature 99 <= 0.0)
    Predict: 0.0
   Else (feature 99 > 0.0)
    Predict: 1.0
  Else (feature 434 > 0.0)
   Predict: 1.0
```

```
17/06/03 16:45:09 INFO DAGScheduler: Job 8 finished: count at /home/lanman/cour
se/dm/treesSparkMLlib.py:26, took 0.089946 s
Test Error = 0.0000%
Learned classification tree model:
DecisionTreeModel classifier of depth 2 with 5 nodes
  If (feature 434 <= 0.0)
   If (feature 100 <= 165.0)
    Predict: 0.0
   Else (feature 100 > 165.0)
    Predict: 1.0
  Else (feature 434 > 0.0)
   Predict: 1.0
```

```
17/06/03 16:44:42 INFO DAGScheduler: Job 8 finished: count at /home/lanman/cour
se/dm/treesSparkMLlib.py:26, took 0.100514 s
Test Error = 2.9412%
Learned classification tree model:
DecisionTreeModel classifier of depth 2 with 5 nodes
  If (feature 406 <= 20.0)
   If (feature 100 <= 165.0)
    Predict: 0.0
   Else (feature 100 > 165.0)
    Predict: 1.0
  Else (feature 406 > 20.0)
   Predict: 1.0
```

```
17/06/03 16:44:13 INFO DAGScheduler: Job 8 finished: count at /home/lanman/cour
se/dm/treesSparkMLlib.py:26, took 0.074256 s
Test Error = 3.8462%
Learned classification tree model:
DecisionTreeModel classifier of depth 2 with 5 nodes
  If (feature 406 <= 72.0)
   If (feature 100 <= 165.0)
    Predict: 0.0
   Else (feature 100 > 165.0)
    Predict: 1.0
  Else (feature 406 > 72.0)
   Predict: 1.0
```

5）Spark ML 中的 DT （ Decision Tree ）算法 (DataFrame-based)

将下面的代码存入DTSparkML.py中，然后提交下面的spark任务：

- cd $SPARK_HOME
- ./bin/spark-submit --master local[8] /home/lanman/course/dm/DTSparkML.py
- 在本地机运行8个线程 Run on local machine using 8 threads

In [ ]:

```
# -*- coding: utf-8 -*-
from pyspark.ml import Pipeline
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.feature import StringIndexer, VectorIndexer
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("DecisionTreeSparkML").getOrCreate()

# Load the data stored in LIBSVM format as a DataFrame.
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.
labelIndexer = StringIndexer(inputCol="label", outputCol="indexedLabel").fit(data)
# Automatically identify categorical features, and index them.
# We specify maxCategories so features with > 4 distinct values are treated as continuous.
featureIndexer =\
    VectorIndexer(inputCol="features", outputCol="indexedFeatures", maxCategories=4).fit(data)

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

# Train a DecisionTree model.
dt = DecisionTreeClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures")

# Chain indexers and tree in a Pipeline
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, dt])

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

# Make predictions.
predictions = model.transform(testData)

# Select example rows to display.
predictions.select("prediction", "indexedLabel", "features").show(5)

# Select (prediction, true label) and compute test error
evaluator = MulticlassClassificationEvaluator(
    labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print "Test Error = %2.4f%% " % ((1.0 - accuracy)*100)

treeModel = model.stages[2]
# summary only
print treeModel

spark.stop()
```

运行 Decision Tree （DataFrame-based）结果

- took 0.050328 s
- Test Error = 0.0000%
- DecisionTreeClassificationModel (uid=DecisionTreeClassifier_4e1ea1e1258574a4a2b7) of depth 2 with 5 nodes

6) Spark MLlib中的 NB （Naive Bayes）算法 （RDD-based）

将下面的代码存入NBSparkMLlib.py中，然后提交下面的spark任务：

- cd $SPARK_HOME
- ./bin/spark-submit --master local[8] /home/lanman/course/dm/NBSparkMLlib.py
- 在本地机运行8个线程 Run on local machine using 8 threads

In [ ]:

```
# -*- coding: utf-8 -*-
from pyspark.ml.classification import NaiveBayes
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("NBSparkMLlib").getOrCreate()

# Load training data
data = spark.read.format("libsvm").load("data/mllib/sample_libsvm_data.txt")# 数据默认存放在 SPA
RK_HOME 目录下

# Split the data into train and test
splits = data.randomSplit([0.6, 0.4], 1234)
train = splits[0]
test = splits[1]

# create the trainer and set its parameters
nb = NaiveBayes(smoothing=1.0, modelType="multinomial")

# train the model
model = nb.fit(train)

# select example rows to display.
predictions = model.transform(test)
predictions.show()

# compute accuracy on the test set
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction",metri
cName="accuracy")
accuracy = evaluator.evaluate(predictions)

print "Test set accuracy = %2.4f%%" % (float(accuracy) * 100)
spark.stop()
```

运行Naive Bayes （RDD-based）的结果

- Test set accuracy = 100.0000% took 0.124705 s

7) Spark ML中的 NB （ Naive Bayes ）算法 （ DataFrame-based ）

将下面的代码存入NBSparkML.py中，然后提交下面的spark任务：

- cd $SPARK_HOME
- ./bin/spark-submit --master local[8] /home/lanman/course/dm/NBSparkML.py
- 在本地机运行8个线程 Run on local machine using 8 threads

In [ ]:

```
# -*- coding: utf-8 -*-
from pyspark.ml.classification import NaiveBayes
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("NaiveBayesSparkML").getOrCreate()

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

# Split the data into train and test
splits = data.randomSplit([0.6, 0.4], 1234)
train = splits[0]
test = splits[1]

# create the trainer and set its parameters
nb = NaiveBayes(smoothing=1.0, modelType="multinomial")

# train the model
model = nb.fit(train)

# select example rows to display.
predictions = model.transform(test)
predictions.show()

# compute accuracy on the test set
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction",
                                              metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print "Test set accuracy = %s" % str(accuracy)

spark.stop()
```

运行Naive Bayes （ DataFrame-based ）的结果

- Test set accuracy = 100.0000% took 0.119124 s

8) Spark MLlib 中的 RandomForest 算法 (RDD-based)

将下面的代码存入RandomForestSparkMLlib.py中，然后提交下面的spark任务：

- cd $SPARK_HOME
- ./bin/spark-submit --master local[8] /home/lanman/course/dm/RandomForestSparkMLlib.py
- 在本地机运行8个线程 Run on local machine using 8 threads

In [ ]:

```
# -*- coding: utf-8 -*-
from pyspark.ml import Pipeline
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.feature import IndexToString, StringIndexer, VectorIndexer
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("RandomForestSparkMLlib").getOrCreate()

# Load and parse the data file, converting it to a DataFrame.
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.
labelIndexer = StringIndexer(inputCol="label", outputCol="indexedLabel").fit(data)

# Automatically identify categorical features, and index them.
# Set maxCategories so features with > 4 distinct values are treated as continuous.
featureIndexer = VectorIndexer(inputCol="features", outputCol="indexedFeatures",
maxCategories=4).fit(data)

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

# Train a RandomForest model.
rf = RandomForestClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures", numTrees=10)

# Convert indexed labels back to original labels.
labelConverter = IndexToString(inputCol="prediction", outputCol="predictedLabel",labels=labelInd
exer.labels)

# Chain indexers and forest in a Pipeline
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, rf, labelConverter])

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

# Make predictions.
predictions = model.transform(testData)

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

# Select (prediction, true label) and compute test error
evaluator = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="predictio
n", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print "Test Error = %2.4f%%" % ((1.0 - accuracy)*100)

#rfModel = model.stages[2]
#print rfModel  # summary only

spark.stop()
```

运行 Random Forest (RDD-based) 的结果

- Test Error = 0.0000% took 0.105005 s
- Test Error = 2.7778% took 0.034569 s

9) Spark ML 中的 RandomForest 算法 (DataFrame-based)

将下面的代码存入RandomForestSparkML.py中，然后提交下面的spark任务：

- cd $SPARK_HOME
- ./bin/spark-submit --master local[8] /home/lanman/course/dm/RandomForestSparkML.py
- 在本地机运行8个线程 Run on local machine using 8 threads

In [ ]:

```
# -*- coding: utf-8 -*-
from pyspark.ml import Pipeline
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.feature import IndexToString, StringIndexer, VectorIndexer
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("RandomForestSparkML").getOrCreate()

# Load and parse the data file, converting it to a DataFrame.
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.
labelIndexer = StringIndexer(inputCol="label", outputCol="indexedLabel").fit(data)

# Automatically identify categorical features, and index them.
# Set maxCategories so features with > 4 distinct values are treated as continuous.
featureIndexer =\
    VectorIndexer(inputCol="features", outputCol="indexedFeatures", maxCategories=4).fit(data)

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

# Train a RandomForest model.
rf = RandomForestClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures", numTrees=10)

# Convert indexed labels back to original labels.
labelConverter = IndexToString(inputCol="prediction", outputCol="predictedLabel",
                               labels=labelIndexer.labels)

# Chain indexers and forest in a Pipeline
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, rf, labelConverter])

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

# Make predictions.
predictions = model.transform(testData)

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

# Select (prediction, true label) and compute test error
evaluator = MulticlassClassificationEvaluator(
    labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print "Test Error = %g" % (1.0 - accuracy)

rfModel = model.stages[2]
print rfModel  # summary only

spark.stop()
```

运行 Random Forest (DataFrame-based) 的结果

- took 0.064762 s
- Test Error = 0
- RandomForestClassificationModel (uid=rfc_4c206ab24a62) with 10 trees

2.2 Spark ML 中的回归算法

1) Spark ML 中的 Linear Regression

SPARK_HOME=/home/lanman/Hadoop/hadoop-2.7.3/spark-2.1.1-bin-hadoop2.7

将下面的代码存入LinearRegressionSparkML.py中，然后提交下面的spark任务：

- cd $SPARK_HOME
- ./bin/spark-submit --master local[8] /home/lanman/course/dm/LinearRegressionSparkML.py
- 在本地机运行8个线程 Run on local machine using 8 threads

In [ ]:

```
# -*- coding: utf-8 -*-
from pyspark.ml.regression import LinearRegression

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("LinearRegressionSparkML").getOrCreate()

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

lr = LinearRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)

# Fit the model
lrModel = lr.fit(training)

# Print the coefficients and intercept for linear regression
print "Coefficients: %s" % str(lrModel.coefficients)
print "Intercept: %s" % str(lrModel.intercept)

# Summarize the model over the training set and print out some metrics
trainingSummary = lrModel.summary
print "numIterations: %d" % trainingSummary.totalIterations
print "objectiveHistory: %s" % str(trainingSummary.objectiveHistory)
trainingSummary.residuals.show()
print "RMSE: %f" % trainingSummary.rootMeanSquaredError
print "r2: %f" % trainingSummary.r2

spark.stop()
```

运行 Linear Regression 的结果

- Coefficients: [0.0,0.322925166774,-0.343854803456,1.91560170235,0.0528805868039,0.76596272046,\ 0.0,-0.151053926692,-0.215879303609,0.220253691888]
- Intercept: 0.159893684424
- numIterations: 7
- objectiveHistory: [0.49999999999999994, 0.4967620357443381, 0.4936361664340463, 0.4936351537897608, 0.4936351214177871, 0.49363512062528014, 0.4936351206216114]
- RMSE: 10.189077
- r2: 0.022861
- residuals:

```
+--------------------+
|           residuals|
+--------------------+
|  -9.889232683103197|
|  0.5533794340053554|
|  -5.204019455758823|
| -20.566686715507508|
|    -9.4497405180564|
|  -6.909112502719486|
|  -10.00431602969873|
|   2.062397807050484|
|  3.1117508432954772|
| -15.893608229419382|
|  -5.036284254673026|
|   6.483215876994333|
|  12.429497299109002|
|  -20.32003219007654|
| -2.0049838218725005|
| -17.867901734183793|
|   7.646455887420495|
| -2.2653482182417406|
|-0.10308920436195645|
|  -1.380034070385301|
+--------------------+
only showing top 20 rows

RMSE: 10.189077
r2: 0.022861
```

2）Spark ML 中的 Generalized Linear Regression

SPARK_HOME=/home/lanman/Hadoop/hadoop-2.7.3/spark-2.1.1-bin-hadoop2.7

将下面的代码存入GeneralLinearRegressionSparkML.py中，然后提交下面的spark任务：

- cd $SPARK_HOME
- ./bin/spark-submit --master local[8] /home/lanman/course/dm/GeneralLinearRegressionSparkML.py
- 在本地机运行8个线程 Run on local machine using 8 threads

In [ ]:

```
# -*- coding: utf-8 -*-
from pyspark.ml.regression import GeneralizedLinearRegression

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("GeneralLinearRegressionSparkML").getOrCreate()

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

glr = GeneralizedLinearRegression(family="gaussian", link="identity", maxIter=10, regParam=0.3)

# Fit the model
model = glr.fit(dataset)

# Print the coefficients and intercept for generalized linear regression model
print("Coefficients: " + str(model.coefficients))
print("Intercept: " + str(model.intercept))

# Summarize the model over the training set and print out some metrics
summary = model.summary
print("Coefficient Standard Errors: " + str(summary.coefficientStandardErrors))
print("T Values: " + str(summary.tValues))
print("P Values: " + str(summary.pValues))
print("Dispersion: " + str(summary.dispersion))
print("Null Deviance: " + str(summary.nullDeviance))
print("Residual Degree Of Freedom Null: " + str(summary.residualDegreeOfFreedomNull))
print("Deviance: " + str(summary.deviance))
print("Residual Degree Of Freedom: " + str(summary.residualDegreeOfFreedom))
print("AIC: " + str(summary.aic))
print("Deviance Residuals: ")
summary.residuals().show()

spark.stop()
```

运行 Generalized Linear Regression 的结果

- Coefficients: [0.0105418280813,0.800325310056,-0.784516554142,2.36798871714,0.501000208986,\ 1.12223511598,-0.292682439862,-0.498371743232,-0.603579718068,0.672555006719]
- Intercept: 0.145921761452
- Coefficient Standard Errors: [0.7950428434287478, 0.8049713176546897, 0.7975916824772489, 0.8312649247659919, 0.7945436200517938, 0.8118992572197593, 0.7919506385542777, 0.7973378214726764, 0.8300714999626418, 0.7771333489686802, 0.463930109648428]
- T Values: [0.013259446542269243, 0.9942283563442594, -0.9836067393599172, 2.848657084633759, 0.6305509179635714, 1.382234441029355, -0.3695715687490668, -0.6250446546128238, -0.7271418403049983, 0.8654306337661122, 0.31453393176593286]
- P Values: [0.989426199114056, 0.32060241580811044, 0.3257943227369877, 0.004575078538306521, 0.5286281628105467, 0.16752945248679119, 0.7118614002322872, 0.5322327097421431, 0.467486325282384, 0.3872259825794293, 0.753249430501097]
- Dispersion: 105.609883568
- Null Deviance: 53229.3654339
- Residual Degree Of Freedom Null: 500
- Deviance: 51748.8429484
- Residual Degree Of Freedom:490
- AIC: 3769.18958718
- Deviance Residuals:

```
+-------------------+
|  devianceResiduals|
+-------------------+
|-10.974359174246889|
| 0.8872320138420559|
| -4.596541837478908|
|-20.411667435019638|
|-10.270419345342642|
|-6.0156058956799905|
|-10.663939415849267|
| 2.1153960525024713|
| 3.9807132379137675|
|-17.225218272069533|
| -4.611647633532147|
| 6.4176669407698546|
| 11.407137945300537|
| -20.70176540467664|
| -2.683748540510967|
|-16.755494794232536|
|  8.154668342638725|
|-1.4355057987358848|
|-0.6435058688185704|
|  -1.13802589316832|
+-------------------+
only showing top 20 rows
```

3）Spark ML 中的 Decision Tree Regression

SPARK_HOME=/home/lanman/Hadoop/hadoop-2.7.3/spark-2.1.1-bin-hadoop2.7

将下面的代码存入DTRegressionSparkML.py中，然后提交下面的spark任务：

- cd $SPARK_HOME
- ./bin/spark-submit --master local[8] /home/lanman/course/dm/DTreeRegressionSparkML.py
- 在本地机运行8个线程 Run on local machine using 8 threads

In [ ]:

```
# -*- coding: utf-8 -*-
from pyspark.ml import Pipeline
from pyspark.ml.regression import DecisionTreeRegressor
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("DecisionTreeRegressionSparkML").getOrCreate()

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

# Automatically identify categorical features, and index them.
# We specify maxCategories so features with > 4 distinct values are treated as continuous.
featureIndexer=VectorIndexer(inputCol="features", outputCol="indexedFeatures",
maxCategories=4).fit(data)

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

# Train a DecisionTree model.
dt = DecisionTreeRegressor(featuresCol="indexedFeatures")

# Chain indexer and tree in a Pipeline
pipeline = Pipeline(stages=[featureIndexer, dt])

# Train model.  This also runs the indexer.
model = pipeline.fit(trainingData)

# Make predictions.
predictions = model.transform(testData)

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

# Select (prediction, true label) and compute test error
evaluator=RegressionEvaluator(labelCol="label", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
print "Root Mean Squared Error (RMSE) on test data = %g" % (rmse)

treeModel = model.stages[1]
# summary only
print treeModel

spark.stop()
```

运行 Decision Tree Regression 的结果

- took 0.052034 s
- Root Mean Squared Error (RMSE) on test data = 0.196116
- DecisionTreeRegressionModel (uid=DecisionTreeRegressor_44c5a6caefcd3d6c38e2) of depth 2 with 5 nodes

2.3 Spark ML 中的 聚类算法

1）Spark ML 中的 KMeans算法

SPARK_HOME=/home/lanman/Hadoop/hadoop-2.7.3/spark-2.1.1-bin-hadoop2.7

将下面的代码存入KMeansSparkML.py中，然后提交下面的spark任务：

- cd $SPARK_HOME
- ./bin/spark-submit --master local[8] /home/lanman/course/dm/KMeansSparkML.py
- 在本地机运行8个线程 Run on local machine using 8 threads

In [ ]:

```
# -*- coding: utf-8 -*-
from pyspark.ml.clustering import KMeans

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("KMeansSparkMLlib").getOrCreate()

# Loads data.
dataset = spark.read.format("libsvm").load("data/mllib/sample_kmeans_data.txt")

# Trains a k-means model.
kmeans = KMeans().setK(2).setSeed(1)
model = kmeans.fit(dataset)

# Evaluate clustering by computing Within Set Sum of Squared Errors.
wssse = model.computeCost(dataset)
print "Within Set Sum of Squared Errors = %s" % str(wssse)

# Shows the result.
centers = model.clusterCenters()
print "Cluster Centers: "
for center in centers:
    print center

spark.stop()
```

运行KMeans算法的结果

- Within Set Sum of Squared Errors = 0.12
- Cluster Centers:
  - [0.1 0.1 0.1]
  - [9.1 9.1 9.1]

2）Spark ML 中的 LDA 算法

SPARK_HOME=/home/lanman/Hadoop/hadoop-2.7.3/spark-2.1.1-bin-hadoop2.7

将下面的代码存入LDASparkML.py中，然后提交下面的spark任务：

- cd $SPARK_HOME
- ./bin/spark-submit --master local[8] /home/lanman/course/dm/LDASparkML.py
- 在本地机运行8个线程 Run on local machine using 8 threads

In [ ]:

```
# -*- coding: utf-8 -*-
from pyspark.ml.clustering import LDA

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("LDASparkMLlib").getOrCreate()

# Loads data.
dataset = spark.read.format("libsvm").load("data/mllib/sample_lda_libsvm_data.txt")

# Trains a LDA model.
lda = LDA(k=10, maxIter=10)
model = lda.fit(dataset)

ll = model.logLikelihood(dataset)
lp = model.logPerplexity(dataset)
print "The lower bound on the log likelihood of the entire corpus: %s" % str(ll)
print "The upper bound bound on perplexity: %s" % str(lp)

# Describe topics.
topics = model.describeTopics(3)
print "The topics described by their top-weighted terms:"
topics.show(truncate=False)

# Shows the result
transformed = model.transform(dataset)
transformed.show(truncate=False)

spark.stop()
```

运行 LDA 算法的结果

- The lower bound on the log likelihood of the entire corpus: -803.7436446
- The upper bound bound on perplexity: 3.09132171537
- The topics described by their top-weighted terms:

topic	termIndices	termWeights
0	[4, 7, 10]	[0.10782285485878747, 0.09748059143552852, 0.0962348773122605]
1	[1, 6, 9]	[0.16755682616392378, 0.14746674856668385, 0.12291623240522744]
2	[1, 3, 9]	[0.1006440767189491, 0.1004423254078914, 0.09911428737697336]
3	[1, 3, 7]	[0.10157583518944054, 0.09974498527193565, 0.09902599515523332]
4	[3, 10, 6]	[0.2377120640896986, 0.11929724807191798, 0.09416811128967989]
5	[8, 5, 7]	[0.10843493099399856, 0.09701504451542989, 0.09334497883908023]
6	[8, 5, 0]	[0.09874157102774704, 0.09654281094035122, 0.09565957008225691]
7	[9, 4, 7]	[0.11252482588278553, 0.09755087991440967, 0.09643430831104527]
8	[4, 1, 2]	[0.10994284274969336, 0.09410689814870361, 0.09374716313351238]
9	[5, 4, 0]	[0.15265940086626775, 0.14015412560969903, 0.13878634715563895]

2.4 Spark ML 中的 关联规则挖掘 算法

1） Spark MLlib 中的 FP-Growth 算法

SPARK_HOME=/home/lanman/Hadoop/hadoop-2.7.3/spark-2.1.1-bin-hadoop2.7

将下面的代码存入KMeansSparkML.py中，然后提交下面的spark任务：

- cd $SPARK_HOME
- ./bin/spark-submit --master local[8] /home/lanman/course/dm/FPGrowthSparkMLlib.py
- 在本地机运行8个线程 Run on local machine using 8 threads

In [ ]:

```
# -*- coding: utf-8 -*-

from pyspark import SparkContext
from pyspark.sql import SQLContext
from pyspark.mllib.fpm import FPGrowth

# 创建sc环境
sc = SparkContext()
sqlContext = SQLContext(sc)

data = sc.textFile("/home/lanman/course/dm/data/sample_fpgrowth.txt")
transactions = data.map(lambda line: line.strip().split(' '))
model = FPGrowth.train(transactions, minSupport=0.2, numPartitions=10)
result = model.freqItemsets().collect()

for fi in result:
        print fi

# 输出结果到文本文件
resultfile = open('/home/lanman/course/dm/data/output-fpgrwothSparkMLlibResult.txt','w')
for fi in result:
    print(fi)
    print >> resultfile, fi

resultfile.close()
```

目前

- SKlearn中没有实现关联规则的算法
- Weka中实现Apriori算法
- Spark ML 中没有实现Apriori算法，但是在RDD-based API的MLlib中有FPGrowth算法

运行FPGrowthSparkMLlib.py代码，得出的结果为项集和频率

运行时间：

- took 0.471368 s
- took 0.468563 s

输出结果到 fpgrwothSparkMLlibResult.txt文件

结论： Try more and update timely！