Big Data Analysis: Assignment 1 Part Two and Three: Spark Scala and PySpark

Baochen Hu
Ning Zhang

June 20, 2015

Zusammenfassung

This part of the assignment aims to practice algorithms in Scala and PySpark with Spark. We used the white vinho verde wine samples data, from the north of Portugal. We separated the dataset into two parts. The first part is our training dataset. The second part is our test dataset for the results.

1 Development Environment

Spark version 1.3.1
sbt

2 Running Introduction

Use sbt build jar package
Command:

```
sbt package
```

Use spark-submit in /spark/bin run the jar file we built in your scala project Example:

```
./spark-submit ../../../../../Users/ningzhang/Desktop/Assignment1/spark_scala_ml/LassoWithSGD/target/scala-2.10/simple-project_2.10-1.0.jar
```

3 Tryout

We will try out the algorithms in Scala and PySpark using Spark MLlib with the following models:

Classification:

```
LogisticRegressionWithLBFGS
LogisticRegressionWithSGD
```

Regression:

```
LinearRegressionWithSGD
RidgeRegressionWithSGD
?LassoWithSGD
```

3.1 LogisticRegressionWithLBFGS

3.1.1 Introduction

Train a classification model for Multinomial/Binary Logistic Regression using Limited-memory BFGS. Standard feature scaling and L2 regularization are used by default. NOTE: Labels used in Logistic Regression should be 0, 1, ..., k - 1 for k classes multi-label classification problem.

3.1.2 Source Code in Scala:

```
/* SimpleApp.scala */
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
import org.apache.spark.mllib.evaluation.MulticlassMetrics
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.classification.{LogisticRegressionWithLBFGS, LogisticRegressionModel}
object SimpleApp {
  def main(args: Array[String]) {
    val csvPath = "/Users/baochenhu/Desktop/spark-1.3.1-bin-hadoop2.4/winequality-white.csv"
    // Should be some file on your system
    val conf = new SparkConf().setAppName("Simple Application")
    val sc = new SparkContext(conf)
    val csv = sc.textFile(csvPath)
    val headerAndRows= csv.map(line=>line.split(";").map(_.trim))
    val header = headerAndRows.first
    val data = headerAndRows.filter(_(0) != header(0))
    val parsedData = data.map{line => LabeledPoint(line(11).toDouble, Vectors.dense(line(0).toDouble,line(1).toDouble))}
    parsedData.cache()
    val splits = parsedData.randomSplit(Array(0.6, 0.4), seed = 11L)
    val training = splits(0).cache()
    val test = splits(1)
    val numIterations = 100
    val model = new LogisticRegressionWithLBFGS().setNumClasses(10).run(training)
    val predictionAndLabels = test.map { case LabeledPoint(label, features) =>
     val prediction = model.predict(features)
           (prediction, label)}


    // Get evaluation metrics.
val metrics = new MulticlassMetrics(predictionAndLabels)
val precision = metrics.precision
println("Precision = " + precision)
  }
}
\subsubsection{Output:}
\begin{lstlisting}
Precision = 0.4532994923857868
```

3.1.3 Source Code in PySpark:

```
from pyspark.mllib.regression import LabeledPoint, LinearRegressionWithSGD
from numpy import array
from pyspark import SparkContext
from pyspark.mllib.classification import LogisticRegressionWithLBFGS
# Load and parse the data
def parsePoint(line):
    values = [float(x) for x in line.split(';')]
    return LabeledPoint(values[11], values[0:10])

```

```

sc = SparkContext("local", "Simple App")
data = sc.textFile("../winequality.csv")
parsedData = data.map(parsePoint)

# Build the model
model = LogisticRegressionWithLBFGS.train(parsedData)

# Evaluating the model on training data
labelsAndPreds = parsedData.map(lambda p: (p.label, model.predict(p.features)))
trainErr = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(parsedData.count())
print("Training Error = " + str(trainErr))
```

3.2 LogisticRegressionWithSGD

3.2.1 Introduction

Train a classification model for Binary Logistic Regression using Stochastic Gradient Descent. By default L2 regularization is used, which can be changed via LogisticRegressionWithSGD.optimizer. NOTE: Labels used in Logistic Regression should be 0, 1, ..., k - 1 for k classes multi-label classification problem. Using LogisticRegressionWithLBFGS is recommended over this.

3.2.2 Source Code:

```
/* SimpleApp.scala */
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.classification.LogisticRegressionWithSGD
object SimpleApp {
  def main(args: Array[String]) {
    val csvPath = "/Users/ningzhang/Desktop/Assignment1/winequality-white.csv" // Should be some file on your system
    val conf = new SparkConf().setAppName("Simple Application")
    val sc = new SparkContext(conf)
    val csv = sc.textFile(csvPath)
    val headerAndRows= csv.map(line=>line.split(";").map(_.trim))
    val header = headerAndRows.first
    val data = headerAndRows.filter(_(0) != header(0))
    val parsedData = data.map{line => LabeledPoint(line(11).toDouble, Vectors.dense(line(0).toDouble,line(1).toDouble))}
    parsedData.cache()
    // Run training algorithm to build the model
        val numIterations = 20
        val model = LogisticRegressionWithSGD.train(parsedData, numIterations)

        // Evaluate model on training examples and compute training error
        val labelAndPreds = parsedData.map { point =>
          val prediction = model.predict(point.features)
          (point.label, prediction)
        }
        val trainErr = labelAndPreds.filter(r => r._1 != r._2).count.toDouble / parsedData.count
        val n1 = labelAndPreds.filter(r => (r._1==1)&&(r._2==1)).count.toDouble
        val n2 = labelAndPreds.filter(r => (r._1==0)&&(r._2==0)).count.toDouble
        val d1 = labelAndPreds.filter(r => (r._1==1)).count.toDouble
        val d2 = labelAndPreds.filter(r => (r._1==0)).count.toDouble
        val sensitivity = n1/d1
        val specificity = n2/d2

        println("\nTraining Error = " + trainErr)
        println("Sensitivity = " + sensitivity)
        println("Specificity = " + specificity)
        println();
  }
}
```

3.2.3 Output:

```
Training Error = 0.461624026696
Sensitivity = 0.48275862069
Specificity = 0.47620692758
```

3.3 LinearRegressionWithSGD

3.3.1 Introduction

Train a linear regression model with no regularization using Stochastic Gradient Descent. This solves the least squares regression formulation f(weights) = 1/n ||A weights-y||2 (which is the mean squared error). Here the data matrix has n rows, and the input RDD holds the set of rows of A, each with its corresponding right hand side label y. See also the documentation for the precise formulation.

3.3.2 Source Code in Scala:

```
/* SimpleApp.scala */
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.regression.LinearRegressionWithSGD
import org.apache.spark.mllib.regression.RidgeRegressionWithSGD
object SimpleApp {
  def main(args: Array[String]) {
    val csvPath = "/Users/ningzhang/Assignment1/winequality-white.csv" // Should be some file on your system
    val conf = new SparkConf().setAppName("Simple Application")
    val sc = new SparkContext(conf)
    val csv = sc.textFile(csvPath)
    val headerAndRows= csv.map(line=>line.split(";").map(_.trim))
    val header = headerAndRows.first
    val data = headerAndRows.filter(_(0) != header(0))
    val parsedData = data.map{line => LabeledPoint(line(11).toDouble, Vectors.dense(line(0).toDouble,line(1).toDouble))}
    parsedData.cache()
    val numIterations = 20
    val model = LinearRegressionWithSGD.train(parsedData, numIterations)
    val valuesAndPreds = parsedData.map { point =>
           val prediction = model.predict(point.features)
            (point.label, prediction)}
     val MSE = valuesAndPreds.map{ case(v, p) => math.pow((v - p), 2)}.reduce(_ + _)/valuesAndPreds.count
     println("LinearRegressionWithSGD training Mean Squared Error = " + MSE)
     println("----------------------------------")
    val model1 = RidgeRegressionWithSGD.train(parsedData, numIterations)
    val valuesAndPreds1 = parsedData.map { point =>
           val prediction = model1.predict(point.features)
            (point.label, prediction)}
     val MSE1 = valuesAndPreds1.map{ case(v, p) => math.pow((v - p), 2)}.reduce(_ + _)/valuesAndPreds1.count
    println("LinearRegressionWithSGD training Mean Squared Error = " + MSE1)
    println("---------------------------------------------")
  }
}
```

3.3.3 Output:

```
LinearRegressionWithSGD training Mean Squared Error = 1.4220919486200828E49
```

3.3.4 Source Code in PySpark:

```
from pyspark.mllib.regression import LabeledPoint, LinearRegressionWithSGD
from numpy import array
from pyspark import SparkContext
# Load and parse the data
def parsePoint(line):
    values = [float(x) for x in line.split(';')]
    return LabeledPoint(values[11], values[0:10])


sc = SparkContext("local", "Simple App")
data = sc.textFile("../winequality.csv")
parsedData = data.map(parsePoint)

# Build the model
model = LinearRegressionWithSGD.train(parsedData)

# Evaluate the model on training data
valuesAndPreds = parsedData.map(lambda p: (p.label, model.predict(p.features)))
MSE = valuesAndPreds.map(lambda (v, p): (v - p)**2).reduce(lambda x, y: x + y) / valuesAndPreds.count()
print("Mean Squared Error = " + str(MSE))
```

3.4 RidgeRegressionWithSGD

3.4.1 Introduction

Train a regression model with L2-regularization using Stochastic Gradient Descent. This solves the l1-regularized least squares regression formulation f(weights) = 1/2n ||A weights-y||2 + regParam/2 ||weights||2 Here the data matrix has n rows, and the input RDD holds the set of rows of A, each with its corresponding right hand side label y. See also the documentation for the precise formulation.

3.4.2 Source Code in Scala:

```
/* SimpleApp.scala */
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.regression.LinearRegressionWithSGD
import org.apache.spark.mllib.regression.RidgeRegressionWithSGD
object SimpleApp {
  def main(args: Array[String]) {
    val csvPath = "/Users/baochenhu/Desktop/spark-1.3.1-bin-hadoop2.4/winequality-white.csv" // Should be some file on your system
    val conf = new SparkConf().setAppName("Simple Application")
    val sc = new SparkContext(conf)
    val csv = sc.textFile(csvPath)
    val headerAndRows= csv.map(line=>line.split(";").map(_.trim))
    val header = headerAndRows.first
    val data = headerAndRows.filter(_(0) != header(0))
    val parsedData = data.map{line => LabeledPoint(line(11).toDouble, Vectors.dense(line(0).toDouble,line(1).toDouble))}
    parsedData.cache()
    val numIterations = 20
    val model1 = RidgeRegressionWithSGD.train(parsedData, numIterations)
    val valuesAndPreds1 = parsedData.map { point =>
           val prediction = model1.predict(point.features)
            (point.label, prediction)}
     val MSE1 = valuesAndPreds1.map{ case(v, p) => math.pow((v - p), 2)}.reduce(_ + _)/valuesAndPreds1.count
     println("RidgeRegressionWithSGD training Mean Squared Error = " + MSE1)
     println("------------------------------------")
  }
}
```

3.4.3 Output:

```
RidgeRegressionWithSGD training Mean Squared Error = 1.4220919486200828E49
```

3.4.4 Source Code in PySpark:

```
from pyspark.mllib.regression import LabeledPoint, RidgeRegressionWithSGD
from numpy import array
from pyspark import SparkContext
from pyspark.mllib.classification import LogisticRegressionWithLBFGS
# Load and parse the data
def parsePoint(line):
    values = [float(x) for x in line.split(';')]
    return LabeledPoint(values[11], values[0:10])


sc = SparkContext("local", "Simple App")
data = sc.textFile("../winequality.csv")
parsedData = data.map(parsePoint)

# Build the model
model = RidgeRegressionWithSGD.train(parsedData)

# Evaluating the model on training data
labelsAndPreds = parsedData.map(lambda p: (p.label, model.predict(p.features)))
trainErr = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(parsedData.count())
print("Training Error = " + str(trainErr))
```

3.5 LassonWithSGD

3.5.1 Introduction

Train a regression model with L1-regularization using Stochastic Gradient Descent. This solves the 1-regularized least squares regression formulation f(weights) = 1/2n A weights-y 2 + regParam weights - 1

Here the data matrix has n rows, and the input RDD holds the set of rows of A, each with its corresponding right hand side label y. See also the documentation for the precise formulation.

3.5.2 Source Code in Scala:

```scala
/* SimpleApp.scala */
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.regression.LinearRegressionWithSGD
import org.apache.spark.mllib.regression.LassoWithSGD
object SimpleApp {
  def main(args: Array[String]) {
    val csvPath = "/Users/baochenhu/Desktop/spark-1.3.1-bin-hadoop2.4/winequality-white.csv" // Should be some file on your system
    val conf = new SparkConf().setAppName("Simple Application")
    val sc = new SparkContext(conf)
    val csv = sc.textFile(csvPath)
    val headerAndRows= csv.map(line=>line.split(";").map(_.trim))
    val header = headerAndRows.first
    val data = headerAndRows.filter(_(0) != header(0))
    val parsedData = data.map{line => LabeledPoint(line(11).toDouble, Vectors.dense(line(0).toDouble,line(1).toDouble))}
    parsedData.cache()
    val numIterations = 20
    val model = LassoWithSGD.train(parsedData, numIterations)
    val valuesAndPreds = parsedData.map { point =>
           val prediction = model.predict(point.features)
            (point.label, prediction)}
     val MSE = valuesAndPreds.map{ case(v, p) => math.pow((v - p), 2)}.reduce(_ + _)/valuesAndPreds.count
     println("LassoWithSGD training Mean Squared Error = " + MSE)
     println("----------------------------------------")

  }
}
```

3.5.3 Output:

```
LassoWithSGD training Mean Squared Error = 1.409246080888596E49
```

3.5.4 Source Code in PySpark:

```python
from pyspark.mllib.regression import LabeledPoint, LassoWithSGD
from numpy import array
from pyspark import SparkContext
from pyspark.mllib.classification import LogisticRegressionWithLBFGS
# Load and parse the data
def parsePoint(line):
    values = [float(x) for x in line.split(';')]
    return LabeledPoint(values[11], values[0:10])


sc = SparkContext("local", "Simple App")
data = sc.textFile("../winequality.csv")
parsedData = data.map(parsePoint)

```

```
# Build the model
model = LassoWithSGD.train(parsedData)

# Evaluating the model on training data
labelsAndPreds = parsedData.map(lambda p: (p.label, model.predict(p.features)))
trainErr = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(parsedData.count())
print("Training Error = " + str(trainErr))
```