Assignment 7 - MLlib

1. Write a standalone application for binary classification. In this application, first load the data from the given input file (`sample_svm_data`) in the course page, and execute the SVM training algorithm on this training data. Then, make predictions with the resulting model to compute the training error.

```scala
import org.apache.spark.SparkContext
import org.apache.spark.mllib.classification.SVMWithSGD
import org.apache.spark.mllib.regression.LabeledPoint

object Classification {
  def main(args: Array[String]) {
    val sc = new SparkContext("local", "Classification", "127.0.0.1",
      List("target/scala-2.10/classification_2.10-1.0.jar"))

    // Load and parse the data file
    val data = sc.textFile("../data/sample_svm_data")
    val parsedData = data.map { line =>
      val parts = line.split(' ')
      LabeledPoint(parts(0).toDouble, parts.tail.map(x => x.toDouble).toArray)
    }

    // Run training algorithm to build the model
    val numIterations = 20
    val model = SVMWithSGD.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
    println("Training Error = " + trainErr)
  }
}

// simple.sbt:
name := "Classification"

version := "1.0"

scalaVersion := "2.10.3"

libraryDependencies ++= Seq(
    "org.apache.spark" %% "spark-core" % "0.9.0-incubating",
    "org.apache.spark" %% "spark-mllib" % "0.9.0-incubating"
)

resolvers += "Akka Repository" at "http://repo.akka.io/releases/"
```

2. Write a standalone application for linear regression. Repeat the steps in question 1 on the given training data set (`lpsa.data`) in the course page.

```scala
import org.apache.spark.SparkContext
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.regression.LinearRegressionWithSGD

object LinearRegression {
  def main(args: Array[String]) {
    val sc = new SparkContext("local", "LinearRegression", "127.0.0.1",
      List("target/scala-2.10/linear-regression_2.10-1.0.jar"))

    // Load and parse the data
    val data = sc.textFile("../data/lpsa.data")
    val parsedData = data.map { line =>
      val parts = line.split(',')
      LabeledPoint(parts(0).toDouble, parts(1).split(' ').map(x => x.toDouble).toArray)
    }

    // Building the model
    val numIterations = 20
    val model = LinearRegressionWithSGD.train(parsedData, numIterations)

    // Evaluate model on training examples and compute training error
    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("training Mean Squared Error = " + MSE)
  }
}
```

3. Write a standalone application for clustering. After loading and parsing the given data (`kmeans_data`), use the KMeans object to cluster the data into two clusters, and compute Within Set Sum of Squared Error (WSSSE).

```scala
import org.apache.spark.SparkContext
import org.apache.spark.mllib.clustering.KMeans

object Clustering {
  def main(args: Array[String]) {
    val sc = new SparkContext("local", "Clustering", "127.0.0.1",
      List("target/scala-2.10/clustering_2.10-1.0.jar"))

    // Load and parse the data
    val data = sc.textFile("../data/kmeans_data")
    val parsedData = data.map( _.split(' ').map(_.toDouble))

    // Cluster the data into two classes using KMeans
    val numIterations = 20
    val numClusters = 2
    val clusters = KMeans.train(parsedData, numClusters, numIterations)

    // Evaluate clustering by computing Within Set Sum of Squared Errors
    val WSSSE = clusters.computeCost(parsedData)

    println("Within Set Sum of Squared Errors = " + WSSSE)
  }
}
```