第六章

MLlib线性回归理论与实战

6.1随机梯度下降算法详解：

道士下山：

如果想一最快的速度下山，就是在最大的坡度位置下降。并且，依次迭代。从而可以最快下山。

数学表达公式：

当对不断求导之后，就可得到公式：

xi

公式中的会向着梯度下降的最快方向减少，因此可以腿的最优解。

xi

从而可以的到Java代码：

|  |
| --- |
| **import** java.util.HashMap; **import** java.util.Map;  **public class** sgd { **static public double** *contemp1*=0,*contemp2*=0.1; **public static double** sgc(**double** x, **double** y){  *contemp1*=*contemp1*-*contemp2*\*((*contemp1*\*x)-y);  **return** *contemp1*;  } **public static void** main(String [] args) {  Map map=**new** HashMap();  **for** (**int** i=0;i<=50;i++){  map.put(i,i\*12);  }  map.forEach(  *sgc*(Map.\_1,Map.\_2));  System.***out***.println(*contemp1*);  } }  \*标注\*：上述代码，有一个问题就是map里面的两个数据没有进行变换。 |

6.2线性回归算法：

就是求公式：的解

我们暂时就只求x2的线性回归方程。

首项第一个就是将数据分隔开。每行第一个数据都为y值，之后的两个值分别为对应的x1，和x2的值。并且我们把两个x的值挡住一个整体，存储在一个Vector的向量里面。并且更具迭代的次数，以及步进的系数的设置，我们可以控制每次迭代过程中模型整体的修正程度。

以下为Java代码：

|  |
| --- |
| **import** org.apache.spark.SparkConf; **import** org.apache.spark.api.java.JavaRDD; **import** org.apache.spark.api.java.JavaSparkContext; **import** org.apache.spark.mllib.linalg.Vectors; **import** org.apache.spark.mllib.regression.LabeledPoint; **import** org.apache.spark.mllib.regression.LinearRegressionModel; **import** org.apache.spark.mllib.regression.LinearRegressionWithSGD;  **public class** LinearRegression {  **public static void** main(String[] args){  SparkConf conf = **new** SparkConf().setAppName(**"JavaLinearRegressionWithSGD"**).setMaster(**"local"**);  JavaSparkContext sc = **new** JavaSparkContext(conf);  *// Load and parse the data* String path = **"E:\\task\\data\\spark\\lr.txt"**;  JavaRDD<String> data = sc.textFile(path);  JavaRDD<LabeledPoint> parsedData = data.map(line -> {  String[] parts = line.split(**"|"**);  String[] features = parts[1].split(**","**);  **double**[] v = **new double**[features.**length**];  **for** (**int** i = 0; i < features.**length** - 1; i++) {  v[i] = Double.*parseDouble*(features[i]);  }  **return new** LabeledPoint(Double.*parseDouble*(parts[0]), Vectors.*dense*(v));  });  parsedData.cache();  LinearRegressionModel model =  LinearRegressionWithSGD.*train*(JavaRDD.*toRDD*(parsedData),100, 0.00000001);  **double** result=model.predict(Vectors.*dense*(2));  System.***out***.println(result);  } } |

6.3线性回归实战

下一个为线性回归实战（只不过是将读的文件里面的数据分割符号改变）

Java代码如下：

|  |
| --- |
| **package** gtl.spark.java.example.C06;  **import** org.apache.spark.SparkConf; **import** org.apache.spark.api.java.JavaPairRDD; **import** org.apache.spark.api.java.JavaRDD; **import** org.apache.spark.api.java.JavaSparkContext; **import** org.apache.spark.mllib.linalg.Vectors; **import** org.apache.spark.mllib.regression.LabeledPoint; **import** org.apache.spark.mllib.regression.LinearRegressionModel; **import** org.apache.spark.mllib.regression.LinearRegressionWithSGD; **import** scala.Tuple2;  **public class** LinearRegression2 {   **public static void** main(String [] args) {   SparkConf conf = **new** SparkConf().setAppName(**"JavaLinearRegressionWithSGD"**).setMaster(**"local"**);  JavaSparkContext sc = **new** JavaSparkContext(conf);  *// Load and parse the data* String path = **"E:\\task\\data\\spark\\lpsa.data"**;  JavaRDD<String> data = sc.textFile(path);  JavaRDD<LabeledPoint> parsedData = data.map(line -> {  String[] parts = line.split(**","**);  String[] features = parts[1].split(**" "**);  **double**[] v = **new double**[features.**length**];  **for** (**int** i = 0; i < features.**length** - 1; i++) {  v[i] = Double.*parseDouble*(features[i]);  }  **return new** LabeledPoint(Double.*parseDouble*(parts[0]), Vectors.*dense*(v));  });  parsedData.cache();  LinearRegressionModel model =  LinearRegressionWithSGD.*train*(JavaRDD.*toRDD*(parsedData),100, 0.00000001);  **double** result=model.predict(Vectors.*dense*(2));  System.***out***.println(result);   }   } |

6.4对拟合曲线的验证

均方误差是衡量“平均误差“的一个方便方法，可以评价数据的变化程度。均方根误差是均方误差的算数平方根。

标准误差定义为各测量值的误差的平方和的平方均值的平方根。设那个测量值的误差为,,……,则这组测量数据的标准误差计算公式为：

=

从而设计的Java代码如下：

|  |
| --- |
| **import** org.apache.spark.SparkConf; **import** org.apache.spark.api.java.JavaPairRDD; **import** org.apache.spark.api.java.JavaRDD; **import** org.apache.spark.api.java.JavaSparkContext; **import** org.apache.spark.mllib.linalg.Vectors; **import** org.apache.spark.mllib.regression.LabeledPoint; **import** org.apache.spark.mllib.regression.LinearRegressionModel; **import** org.apache.spark.mllib.regression.LinearRegressionWithSGD; **import** scala.Tuple2;  **public class** MSE {  **public static void** main(String[] args){   SparkConf conf = **new** SparkConf().setAppName(**"JavaLinearRegressionWithSGD"**).setMaster(**"local"**);  JavaSparkContext sc = **new** JavaSparkContext(conf);   *// Load and parse the data* String path = **"E:\\task\\data\\spark\\D06\\lpsa.data"**;  JavaRDD<String> data = sc.textFile(path);  JavaRDD<LabeledPoint> parsedData = data.map(line -> {  String[] parts = line.split(**","**);  String[] features = parts[1].split(**" "**);  **double**[] v = **new double**[features.**length**];  **for** (**int** i = 0; i < features.**length** - 1; i++) {  v[i] = Double.*parseDouble*(features[i]);  }  **return new** LabeledPoint(Double.*parseDouble*(parts[0]), Vectors.*dense*(v));  });  parsedData.cache();   *// Building the model* **int** numIterations = 100;  **double** stepSize = 0.00000001;  LinearRegressionModel model =  LinearRegressionWithSGD.*train*(JavaRDD.*toRDD*(parsedData), numIterations, stepSize);   *// Evaluate model on training examples and compute training error* JavaPairRDD<Double, Double> valuesAndPreds = parsedData.mapToPair(point ->  **new** Tuple2<>(model.predict(point.features()), point.label()));   **double** MSE = valuesAndPreds.mapToDouble(pair -> {  **double** diff = pair.\_1() - pair.\_2();  **return** diff \* diff;  }).mean();  System.***out***.println(**"training Mean Squared Error = "** + MSE);  sc.stop();  } } |