  
第10章 大數據分析範例－葡萄酒分析



### 讀入資料檔

|  |  |
| --- | --- |
| 指令 | 說明 |
| 進入Ubuntu環境 | |
| start-all.sh | 啟動hadoop，若已經啟動，則不需要再執行此指令 |
| cd ~ | 回到家目錄 |
| wget https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-white.csv | 下載白葡萄酒資料集 |
| hdfs dfs -mkdir /wine | 在hdfs上建一目錄 |
| hdfs dfs -put ~/winequality-white.csv /wine | 將文章上傳hdfs |
| hdfs dfs -ls /wine | 查看hdfs資料 |
| 進入spark-shell環境，開始處理 | |
| spark-shell | 進入spark-shell |
| 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.feature.StandardScaler | 匯入四個必要的程式庫 |
| val rawData=sc.textFile("hdfs://master:9000/wine/winequality-white.csv")  val wineStringRDD=rawData.map(line => line.split(";")).mapPartitionsWithIndex { (index,lines) => if (index==0) lines.drop(1) else lines }  val wineRDD=wineStringRDD.map(x=>x.map(x=>x.toDouble))  wineRDD.count  wineRDD.first | 讀入文本資料，去除首行（欄位名稱），並轉成浮點數 |
| val colNameMap=Map(  0->"固定的酸度",  1->"揮發性酸度",  2->"檸檬酸",  3->"殘糖",  4->"氯化物",  5->"游離二氧化硫",  6->"二氧化硫總量",  7->"密度",  8->"ｐＨ值",  9->"硫酸鹽",  10->"酒精",  11->"品質"  ) | 建立欄位對照表 |

### 基本統計資料

|  |  |
| --- | --- |
| 指令 | 說明 |
| def printStats(rdd: org.apache.spark.rdd.RDD[Array[Double]]) {  for(i<-0 until rdd.first.length) {  val name=colNameMap(i)  val s=rdd.map(x=>x(i)).stats  val sp="".padTo(7-name.length,"　").mkString("")  println(f"$name$sp$s")  }  }  printStats(wineRDD) | 列印資料集的基本統計值 |

### LabeledPoint

|  |  |
| --- | --- |
| 指令 | 說明 |
| val wineLP=wineRDD.map(line=>new LabeledPoint(line.last, Vectors.dense(line.init)))  wineLP.first  wineLP.first.label  wineLP.first.features | 建立 LabeledPoint 資料類別 |

### 資料標準化（standardize）

|  |  |
| --- | --- |
| 指令 | 說明 |
| val wineScaler = new StandardScaler(withMean = true, withStd = true).fit(wineLP.map(x=>x.features)) | 建立StandardScaler物件，配適wineLP的尺度 |
| val scaledWineLP=wineLP.map(x => new LabeledPoint(x.label, wineScaler.transform(x.features)))  printStats(scaledWineLP.map(x=>x.features.toArray)) | 進行資料標準化 |

### 切割資料集

|  |  |
| --- | --- |
| 指令 | 說明 |
| val wineSplit = scaledWineLP.randomSplit(Array(0.7, 0.2, 0.1))  val wineTrainSet = wineSplit(0)  val wineValidSet = wineSplit(1)  val wineTestSet = wineSplit(2)  wineTrainSet.count  wineValidSet.count  wineTestSet.count  wineTrainSet.cache | 切割資料集 |

### 模型訓練

|  |  |
| --- | --- |
| 指令 | 說明 |
| val stepSize=0.1  val numOfIter = 1000  val wineModelInst = new LinearRegressionWithSGD().setIntercept(true)  wineModelInst.optimizer.setNumIterations(numOfIter).setStepSize(stepSize)  val wineModel=wineModelInst.run(wineTrainSet) | 以步進0.1和迭代次數1000訓練模型 |
| def printWeights(w: org.apache.spark.mllib.linalg.Vector) {  val pw=colNameMap.toArray.sortBy(x=>x.\_1).map(x=>x.\_2).zip(w.toArray).sortBy(x=>x.\_2).reverse  pw.foreach{ case (name, wgt) =>  val sp="".padTo(7-name.length,"　").mkString("")  println(f"$name$sp$wgt")  }  }  printWeights(wineModel.weights) | 列印各特徵權重的副程式 |

### 預測與模式評估

|  |  |
| --- | --- |
| 指令 | 說明 |
| val winePred=wineModel.predict(wineValidSet.map(x=>x.features))  winePred.first  wineValidSet.first | 求取預測值 |
| def getMSE(model: org.apache.spark.mllib.regression.LinearRegressionModel, dataset: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint])={  val wineReal=dataset.map(x=>x.label) // 取得真實評分  val winePred=model.predict(dataset.map(x=>x.features)) // 預測評分  val realWithPred=wineReal.zip(winePred) // 真實與預測分數合併  val MSE=realWithPred.map{case (real, pred)=>math.pow(real-pred,2)}.mean()  MSE // 最後結果MSE  }  val MSE=getMSE(wineModel, wineValidSet)  println("MSE: "+ MSE) | 計算MSE的副程式，輸入一個已經訓練好的模型和資料集（通常是驗證資料集） |

### 最佳化參數的最佳組合

|  |  |
| --- | --- |
| 指令 | 說明 |
| def modelAndMSE(stepSize:Double, numOfIter:Int, dataset: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint]) {  // 訓練模型  val wineModelInst = new LinearRegressionWithSGD().setIntercept(true)  wineModelInst.optimizer.setNumIterations(numOfIter).setStepSize(stepSize)  val wineModel=wineModelInst.run(wineTrainSet)  // 求預測值  val winePred=wineModel.predict(dataset.map(x=>x.features))  val wineReal=dataset.map(x=>x.label)  // 計算MSE  val MSE=getMSE(wineModel, dataset)  println("MSE: "+ MSE+"<=="+stepSize+"/"+numOfIter)  } | 訓練某一參數組合（步進+迭代次數）的模型，並計算MSE |
| for(i<-Array(0.1,0.5,1,2,3,4,4.5,5); j<-Array(100,500,1000)) {  modelAndMSE(i,j,wineValidSet)  } | 求取各超參數組合的MSE |
| val stepSize=4  val numOfIter=100  val wineModelInst = new LinearRegressionWithSGD().setIntercept(true)  wineModelInst.optimizer.setNumIterations(numOfIter).setStepSize(stepSize)  val wineModel=wineModelInst.run(wineTrainSet) | 使用最佳超操數組合訓練模型 |
| val MSE=getMSE(wineModel ,wineTestSet)  println("MSE: "+ MSE) | 求最佳模型的MSE |
| printWeights(wineModel.weights) | 列印各特徵變數權重 |