# 实验三 Spark机器学习

## 1. Spark-shell常用指令

### spark-shell

登录服务器，输入spark-shell

```
2021214316@thumm01:~$ spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLeve
l(newLevel).
21/11/16 19:14:46 WARN Utils: Service 'SparkUI' could not bind on port 4040. Att
empting port 4041.
21/11/16 19:14:46 WARN Utils: Service 'SparkUI' could not bind on port 4041. Att
empting port 4042.
21/11/16 19:14:46 WARN Utils: Service 'SparkUI' could not bind on port 4042. Att
empting port 4043.
21/11/16 19:14:46 WARN Utils: Service 'SparkUI' could not bind on port 4043. Att
empting port 4044.
```

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.4
      /_/

Using Scala version 2.12.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_221)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

输入:help

```
scala> :help
All commands can be abbreviated, e.g., :he instead of :help.
:completions <string>    output completions for the given string
:edit <id>|<line>        edit history
:help [command]          print this summary or command-specific help
:history [num]           show the history (optional num is commands to show)
:h? <string>             search the history
:imports [name name ...] show import history, identifying sources of names
:implicits [-v]          show the implicits in scope
:javap <path|class>      disassemble a file or class name
:line <id>|<line>        place line(s) at the end of history
```

使用:load打印Hello Word，注意这里path是相对地址

```
scala> :load helloworld.scala
Loading helloworld.scala...
HelloWorld!
```

## 2. 使用Spark进行词频统计

实验所用数据集为WMT16-newstest2016en.txt，首先将其传入Hadoop文件系统。

```
2021214316@thumm01:~/EXP3$ hadoop fs -put newstest2016en.txt /dsjxtjc/2021214316
/
2021-11-17 13:29:09,503 INFO sasl.SaslDataTransferClient: SASL encryption trust
check: localHostTrusted = false, remoteHostTrusted = false
2021214316@thumm01:~/EXP3$ hadoop fs -ls /dsjxtjc/2021214316
Found 2 items
-rw-r--r--   3 2021214316 dsjxtjc     337711 2021-11-17 13:29 /dsjxtjc/202121431
6/newstest2016en.txt
-rw-r--r--   3 2021214316 dsjxtjc         13 2021-10-19 20:18 /dsjxtjc/202121431
6/test.txt
```

进入spark- shell

```
2021214316@thumm01:~/EXP3$ spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLeve
l(newLevel).
Spark Context Web UI is available at Spark Master Public URL
Spark context available as 'sc' (master = spark://thumm01:7077, app id = app-202
11117133103-0004).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.4
      /_/

Using Scala version 2.12.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_221)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

加载数据集

```
scala> val words = sc.textFile("/dsjxtjc/2021214316/newstest2016en.txt")
words: org.apache.spark.rdd.RDD[String] = /dsjxtjc/2021214316/newstest2016en.txt
 MapPartitionsRDD[1] at textFile at <console>:24
```

查看第一行&查看行数

```
scala> words.first()
res0: String = Obama receives Netanyahu

scala> words.count()
res1: Long = 2999
```

统计词频并保存

```
scala> val result = words.flatMap(l => l.split(" ")).map(w => (w, 1)).reduceByKe
y(_ + _)
result: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[4] at reduceByKey
at <console>:25

scala> result.first()
res2: (String, Int) = (Unfortunately,,1)

scala> result.saveAsTextFile("/dsjxtjc/2021214316/wc_output")
```

打开保存后的hdfs文件，部分结果如下

```
(crown,2)
(midway,1)
(statement,16)
(explanation,,1)
(higher.,1)
(singing,2)
(Statistics,1)
(Stadler.,2)
(7624,1)
(Watch,1)
(Hand,1)
(rip,1)
(Fred,1)
(healthy:,1)
(always,16)
(Foundation,1)
(cash,,1)
(Kurdistan,,1)
(lower,4)
(Smalling.,1)
((2700,1)
(comment,2)
(Luke.,1)
(Müllers,3)
```

## Bonus

为方便与其他词频统计函数对比，先记录map + reduceByKey方法的时间

```
import java.util.Date
words: org.apache.spark.rdd.RDD[String] = /dsjxtjc/2021214316/newstest2016en.txt
 MapPartitionsRDD[18] at textFile at bonus_1.scala:50
t1: Long = 1637133063006
result: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[21] at reduceByKey
 at bonus_1.scala:51
t2: Long = 1637133063400
394
```

法1:使用countByValue代替map + reduceByKey

代码如下

```
import java.util.Date

val words = sc.textFile("/dsjxtjc/2021214316/newstest2016en.txt")
var t1 = new Date().getTime
val result = words.flatMap(l => l.split(" ")).countByValue()
var t2 = new Date().getTime
println(t2-t1)
```

实验结果：耗时713

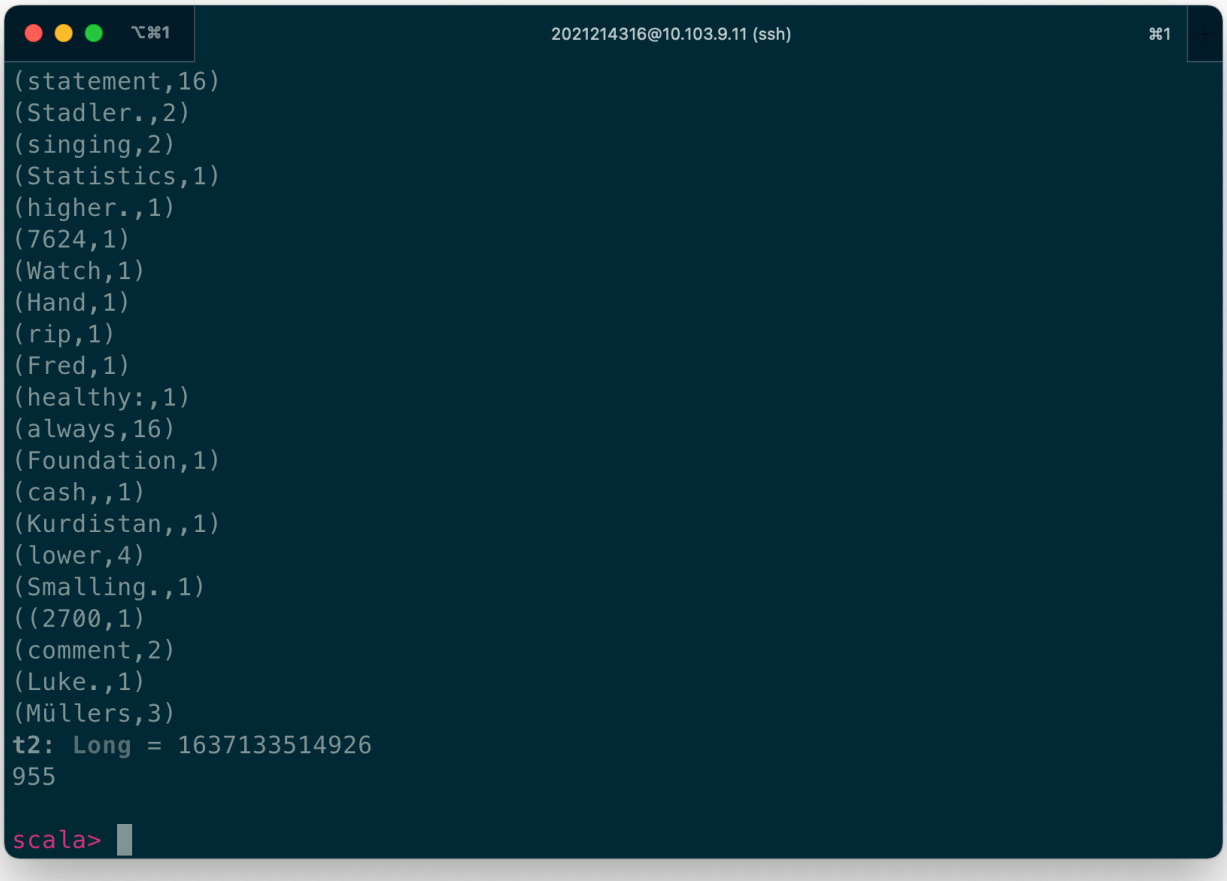
```
scala> :load bonus_1.scala
Loading bonus_1.scala...
import java.util.Date
words: org.apache.spark.rdd.RDD[String] = /dsjxtjc/2021214316/newstest2016en.txt
 MapPartitionsRDD[29] at textFile at bonus_1.scala:54
t1: Long = 1637133091543
result: scala.collection.Map[String,Long] = Map(specifically, -> 2, Agency, -> 1
, come? -> 1, incident -> 4, serious -> 6, ninth. -> 1, brink -> 2, 9/11 -> 2, y
outhful -> 1, comply -> 1, breaks -> 2, checkpoints, -> 1, precious -> 1, E -> 2
, sectors -> 1, dolls. -> 1, thunderous -> 1, Washington, -> 3, embedded -> 1, E
U-sceptic -> 1, Grolsch. -> 1, lover -> 1, lead. -> 1, human, -> 1, Time," -> 1,
 terrible -> 3, EU. -> 2, rate -> 33, pepper -> 3, 2014 -> 3, 5-4. -> 1, rivals"
 -> 1, 45 -> 5, spokeswoman -> 1, submitted -> 2, Mosque, -> 2, League: -> 1, Wh
itman -> 1, everyone, -> 1, KfW -> 1, Cycles -> 1, purchasing -> 1, Hall, -> 1,
30,000 -> 5, WatchOS -> 1, snowball -> 2, looks -> 3, Steigenberger. -> 1, site,
 -> 2, 4th -> 1, "tellingly -> 1, snapped -> 1, Kyr...
t2: Long = 1637133092256
713
```

法二：方法四:groupByKey+map

```scala
import java.util.Date

val config: SparkConf = new SparkConf().setMaster("local[*]").setAppName("WordCount4")
val sc: SparkContext = new SparkContext(config)
val lines: RDD[String] = sc.textFile("/dsjxtjc/2021214316/newstest2016en.txt")
var t1 = new Date().getTime
val groupByKeyRDD: RDD[(String, Iterable[Int])] = lines.flatMap(_.split(" ")).map((_,
1)).groupByKey()
groupByKeyRDD.map(tuple => {
    (tuple._1, tuple._2.sum)
}).collect().foreach(println)
var t2 = new Date().getTime
println(t2-t1)
```

实验结果：耗时955




# 3 使用Spark 计算均值与方差

使用的数据集与实验二相同，此处不再赘述生成方法。

将数据集上传到HDFS

```
2021214316@thumm01:~/EXP3$ hadoop fs -put data.txt /dsjxtjc/2021214316
2021-11-17 15:25:43,461 INFO sasl.SaslDataTransferClient: SASL encryption trust
check: localHostTrusted = false, remoteHostTrusted = false
2021214316@thumm01:~/EXP3$ hadoop fs -tail /dsjxtjc/2021214316/data.txt
2021-11-17 15:25:58,067 INFO sasl.SaslDataTransferClient: SASL encryption trust
check: localHostTrusted = false, remoteHostTrusted = false
4
396
753
256
581
963
817
```

运行spark-shell, 从HDFS 加载number.txt：

```
scala> val numbers = sc.textFile("/dsjxtjc/2021214316/data.txt")
numbers: org.apache.spark.rdd.RDD[String] = /dsjxtjc/2021214316/data.txt MapPart
itionsRDD[1] at textFile at <console>:24
```

加载的数据为字符串形式，需要转成数值型，这里转double型：

```
scala> val numbers_double = numbers.map(num => num.toDouble)
numbers_double: org.apache.spark.rdd.RDD[Double] = MapPartitionsRDD[2] at map at
 <console>:25
```

统计数字个数：

```
scala> val n_num = numbers_double.count()
n_num: Long = 10000
```

计算均值：

```
scala> val mean = numbers_double.reduce(_ + _) / n_num
mean: Double = 500.291
```

计算方差：

```
scala> val variance = numbers_double.map(num => num - mean).map(num => num*num).
reduce(_ + _) / n_num
variance: Double = 84457.06891900033
```

计算标准差：

```
scala> import scala.math._
import scala.math._

scala> val std = sqrt(variance)
std: Double = 290.61498398912664
```

# 4 Spark 机器学习

本题实现了向量表示的2元线性回归。

1. 生成数据集

   每条数据记录包括三个浮点数，分别对应2个特征值和1个标签值，生成1000条记录。

   注：由于scala代码中直接从本地目录调用，无需先传入hadoop中。

```
with open('/Users/xueyuan/Desktop/dataset.txt', 'w') as f:
    for _ in range(1000):
        feature1 = random.random()
        feature2 = random.random()
        lb = 2*feature1 + feature2 + random.random()
        to_write = str(feature1)+' '+str(feature2)+','+str(lb)+'\n'
        f.write(to_write)
f.close()
```

dataset.txt

```
0.3415182213838093 0.35179831513680426,1.4948002226758468
0.4723011225413607 0.6346472826653128,1.8936213086058071
0.8113059644056223 0.23483078377630484,2.472350374098316
0.8759643325679511 0.6718339802978842,2.4648464749446126
0.7185131706088501 0.5003993233202363,2.1404636916601403
0.4079231242367928 0.5088874741667805,2.31251539939192
0.45603183992735585 0.8595775842127462,1.7981492611728487
0.5616993551371298 0.6681570981851424,2.241650745745093
0.4725960431768399 0.8612454276124676,2.7095290001640846
0.1690160316107634 0.9380067258430941,1.7629248256514902
0.36237442378789786 0.2004081036582347,0.9771232755398995
0.6043644252638527 0.979678356209612,3.0053217748010264
0.8195698168755277 0.4193582040983811,2.356746280676777
0.3480110793692578 0.9770305325745342,2.5114288320383373
0.057500897264758155 0.5397963070002908,1.095727049963917
0.25665593915239926 0.495977947400411,1.1184941507793158
0.9921783925595056 0.7393613774447655,3.125592638484401
0.6621014554269143 0.07747581081701627,2.1719683800954104
0.26773639096598667 0.4466579178966499,1.1324599527248225
0.46491369284637063 0.9849079915279046,1.9754715231166875
0.5423561595235035 0.8072576251428074,2.2504253605387046
0.31325329504649135 0.3289483883073592,1.735211306458261
0.644165045272218 0.754748958513933,2.500379104368462
0.16081112385257257 0.5765181980749664,1.5726964989749166
0.0006557017494720485 0.4449267873387529,0.7456926975262103
0.4838277759209272 0.5928880192331658,2.409716718039156
0.3958434221550893 0.8604352961240781,2.6175872735744177
```

2. 线性回归的scala实现

   建立矩阵类，实现矩阵间的运算，此处仅列出部分代码。

```
    def +(a:Matrix):Matrix = {
        if(this.rownum != a.rownum || this.colnum != a.colnum){
            val data:ArrayBuffer[Double] = ArrayBuffer()
            return new Matrix(data.toArray,this.rownum)
        }else{
            val data:ArrayBuffer[Double] = ArrayBuffer()
```

```
            for(i <- 0 until this.rownum){
                for(j <- 0 until this.colnum){
                    data += this.matrix(i)(j) + a.matrix(i)(j)
                }
            }
            return new Matrix(data.toArray,this.rownum)
        }
    }

    def -(a:Matrix):Matrix = {
        if(this.rownum != a.rownum || this.colnum != a.colnum){
            val data:ArrayBuffer[Double] = ArrayBuffer()
            return new Matrix(data.toArray,this.rownum)
        }else{
            val data:ArrayBuffer[Double] = ArrayBuffer()
            for(i <- 0 until this.rownum){
                for(j <- 0 until this.colnum){
                    data += this.matrix(i)(j) - a.matrix(i)(j)
                }
            }
            return new Matrix(data.toArray,this.rownum)
        }
    }

    def *(a:Matrix):Matrix = {
        if(this.colnum != a.rownum){
            val data:ArrayBuffer[Double] = ArrayBuffer()
            return new Matrix(data.toArray,this.rownum)
        }else{
            val data:ArrayBuffer[Double] = ArrayBuffer()
            for(i <- 0 until this.rownum){
                for(j <- 0 until a.colnum){
                    var num = 0.0
                    for(k <- 0 until this.colnum){
                        num += this.matrix(i)(k) * a.matrix(k)(j)
                    }
                data += num
                }
            }
        return new Matrix(data.toArray,this.rownum)
        }
    }
    def transpose():Matrix = {
        val transposeMatrix = for (i <- Array.range(0,colnum)) yield {
             for (rowArray <- this.matrix) yield rowArray(i)
            }
        return new Matrix(transposeMatrix.flatten,colnum)
    }
```

线性回归实现代码

```
var x = new Array[Double](3000)
var y = new Array[Double](1000)

val lines =
Source.fromFile("/home/dsjxtjc/2021214316/EXP3/Task5/dataset.txt").getLines

var i = 0
for (line <- lines){
    val tmpx =line.split(',')(0)
    y(i) = line.split(',')(1).toDouble
    for (j <- 0 to 2)
        x(i*3+j) = tmpx(j).toDouble
    i = i + 1
}

var matX = new Matrix(x,1000)
var matY = new Matrix(y,1000)
val w = new Array[Double](3)
var matW = new Matrix(w,3)

val lr = 0.0000001
val epoch = 50

for (j <- 1 to epoch){
    val maty:Matrix = matX * matW - matY
    val g:Matrix = matX.transpose * maty
    val loss = maty.transpose * maty * (1.0 / (2 * 1000))
    matW = matW - g * lr
    println(loss)
}
```

在spark-shell中运行该脚本

```
scala> :load matrix.scala
Loading matrix.scala...
import scala.collection.mutable.ArrayBuffer
import scala.io.Source
defined class Matrix
```

由于每次新建变量都会输出，此处省略初始化的变量，仅展示loss变化情况，可以看到是在迭代中不断下降的。

由 Xnip 截图

```
lr: Double = 1.0E-7
epoch: Int = 50
2.2315724206100653
0.38841654919576607
0.24114370878963584
```

0.22925168326523931
0.22816711414716
0.22794543250983887
0.22779277304638682
0.22764576212710094
0.22749934055512422
0.2273531043176129
0.22720702102063292
0.22706108793636928
0.22691530470536864
0.22676967115738783
0.2266241871374336
0.22647885249187458
0.22633366706733404
0.22618863071059986
0.22604374326861765
0.22589900458849096
0.22575441451747977
0.22560997290300114
0.22546567959262795
0.22532153443409078
0.22517753727527645
0.22503368796422735
0.22488998634914165
0.22474643227837449
0.22460302560043566
0.22445976616399074
0.22431665381786156
0.22417368841102359
0.22403086979260908
0.2238881978119038
0.22374567231834908
0.2236032931615396
0.22346106019122766
0.22331897325731506
0.2231770322098621
0.22303523689908009
0.2228935871753355
0.22275208288914794
0.22261072389119155
0.22246951003229243
0.22232844116342987
0.22218751713573695
0.22204673780049977
0.2219061030091564

```
0.22176561261329733
0.22162526646466615

scala>
```