目录

[一、安装R语言支持。 2](#_Toc441225947)

[二、SparkR环境搭建。 2](#_Toc441225948)

[三、简单尝试wordcount。 6](#_Toc441225949)

[四、读取数据源的方式。 6](#_Toc441225950)

[五、利用DataFrames读取其他格式文件。 8](#_Toc441225951)

[六、利用DataFrame进行查询、分组、排序、聚合、过滤、去重等操作。 9](#_Toc441225952)

[七、利用DataFrames注册临时表，使用sql查询数据。 12](#_Toc441225953)

[八、改写线上首页抽屉视频点击报表。 12](#_Toc441225954)

[九、与mllib相结合，目前支持拟合广义线性模型。 15](#_Toc441225955)

[十、SparkR 1.6新添加内容。 17](#_Toc441225956)

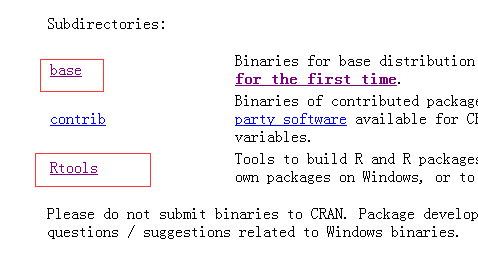
[十一、优化建议。 17](#_Toc441225957)

[十二、遇到的问题。 17](#_Toc441225958)

[十三、操作过程中经常用的R函数。 18](#_Toc441225959)

## 一、安装R语言支持。

安装R base和R tools，从R官网(<https://www.r-project.org/>)分别下载安装，如图：



分别将R base和R tools的bin目录配置到环境变量中。

## 二、SparkR环境搭建。

方式一：

在R命令行中操作

> install.packages('rJava')

#可能提醒未安装RCurl，根据提示信息先install.packages(‘RCurl’)

> install.packages('devtools')

>library(devtools)

>install\_github("amplab-extras/SparkR-pkg", ref= "sparkr-sql" , subdir="pkg")

日志信息：

Downloading GitHub repo amplab-extras/SparkR-pkg@master

Installing SparkR

"D:/tools/R-3.1.3/bin/i386/R" --no-site-file --no-environ --no-save --no-restore CMD INSTALL \

"C:/Users/dell/AppData/Local/Temp/RtmpY7BoCb/devtools259833e75fb5/amplab-extras-SparkR-pkg-e532627/pkg" \

--library="D:/tools/R-3.1.3/library" --install-tests

\* installing \*source\* package 'SparkR' ...

\*\* libs

running 'src/Makefile.win' ...

mvn.bat -Dhadoop.version=1.0.4 -Dspark.version=1.1.0 -Dyarn.version=2.4.0 -DskipTests clean package shade:shade

[INFO] Scanning for projects...

……

[WARNING] We have a duplicate org/objenesis/strategy/StdInstantiatorStrategy.class in E:\maven\repository\org\objenesis\objenesis

\1.2\objenesis-1.2.jar

[INFO] Attaching shaded artifact.

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 02:34 min

[INFO] Finished at: 2015-12-15T13:01:01+08:00

[INFO] Final Memory: 31M/196M

[INFO] ------------------------------------------------------------------------

cp -f target/sparkr-0.1-assembly.jar ../inst/sparkr-assembly-0.1.jar

R CMD SHLIB -o SparkR.dll string\_hash\_code.c

gcc -m32 -I"D:/tools/R-31~1.3/include" -DNDEBUG -I"d:/RCompile/CRANpkg/extralibs64/local/include" -O3 -Wall -std=gnu99 -mtune=core2

-c string\_hash\_code.c -o string\_hash\_code.o

gcc -m32 -shared -s -static-libgcc -o SparkR.dll tmp.def string\_hash\_code.o -Ld:/RCompile/CRANpkg/extralibs64/local/lib/i386 -

Ld:/RCompile/CRANpkg/extralibs64/local/lib -LD:/tools/R-31~1.3/bin/i386 -lR

installing to D:/tools/R-3.1.3/library/SparkR/libs/i386

\*\* R

\*\* inst

\*\* tests

\*\* preparing package for lazy loading

Creating a generic function for 'lapply' from package 'base' in package 'SparkR'

Creating a generic function for 'Filter' from package 'base' in package 'SparkR'

\*\* help

\*\*\* installing help indices

\*\* building package indices

\*\* testing if installed package can be loaded

\* DONE (SparkR)

> library(SparkR)

[SparkR] Initializing with classpath D:/tools/R-3.1.3/library/SparkR/sparkr-assembly-0.1.jar

方式二：

从spark官网下载源码和编译好的代码各一份到本地

Spark-1.5.2.tgz

Spark-1.5.1-bin-hadoop2.6.tgz

解压，将SPARK\_HOME设置成~\spark-1.5.1-bin-hadoop2.6此目录

进入~\spark-1.5.2\R目录下，执行编译命令：

install-dev.bat USE\_MAVEN=1 SPARK\_VERSION=1.5.2 HADOOP\_VERSION=2.6.0

#默认采用SBT编译，USE\_MAVEN=1代表使用MAVEN进行编译。

编译完成将在~\spark-1.5.2\R目录下生成lib目录，将lib目录拷贝到：

~\spark-1.5.1-bin-hadoop2.6\R\目录下

打开Rgui

>Sys.setenv(SPARK\_HOME = “E:/spark/source/spark-1.5.1-bin-hadoop2.6”)

>.libPaths(c(file.path(Sys.getenv("SPARK\_HOME"), "R", "lib"), .libPaths()))

#即在library(SparkR)时加载使用之前编译好的SparkR库。

> .libPaths()

[1] "E:/spark/source/spark-1.5.2-bin-hadoop2.6/R/lib" #加载此目录的库文件

> library(SparkR)

> sc <- sparkR.init(master='local')

Launching java with spark-submit command E:/spark/source/spark-1.5.1-bin-hadoop2.6/bin/spark-submit.cmd sparkr-shell C:\Users\ADMINI~1\AppData\Local\Temp\Rtmp8gxVcw\backend\_port24b44dbaebf

#出现以上信息说明R调用SparkR程序环境搭建成功。

方式三：

直接编译整个spark源码，编译支持sparkr，例如用maven编译：

mvn -Pyarn -Dhadoop.version=2.7.1 -Dyarn.version=2.4.0 -Phive -Psparkr -Pscala-2.10 -Dscala.version=2.10.6 –Dskip Tests clean package

编译Spark源码建议：

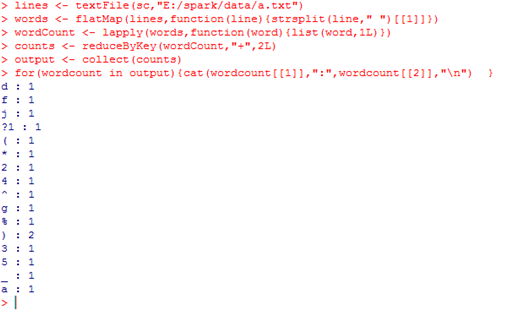
无论在windows还是linux下进行编译，建议maven选用3.3.3及以上，内存保留2G以上，网络要好。整个编译过程要2小时以上。生成部署包需要执行

make-distribution.sh脚本，生成部署包过程也大概需要2小时以上了，简单调试代码不需要生成部署包，不需要部署，可直接进入编译好的bin目录敲击sparkR即可。

启动方式：通过前两种方式安装，先进入R CMD 然后library(SparkR)。

通过编译整个spark源码的方式安装，可以在cmd中敲sparkR直接进入sparkR命令行(已将SparkContext和SparkRSQL初始化完成)。

## 三、简单尝试wordcount。



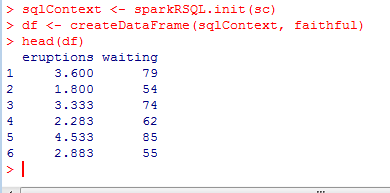
在R里1L 是int型，1是num类型。

lapply是应用在列表或向量上的函数，在这里为了构造成key/value形式。

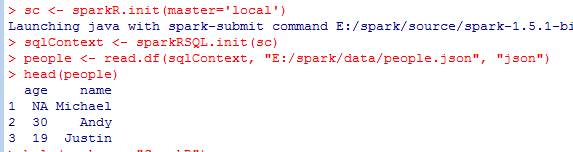
## 四、读取数据源的方式。

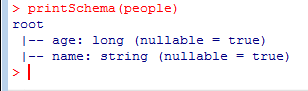
Faithful是 old faithful火山爆发的数据，其中eruption是火山爆发的持续时间，waiting是持续间隔。

(1)从R data frame读取数据集faithful(faithful是R当中已有的数据集，在这里相当于一个本地data frame 或者可以自己创建数据集,例如data.frame(name=c(“A”,”B”,”C”),age=c(18,19,20)))：

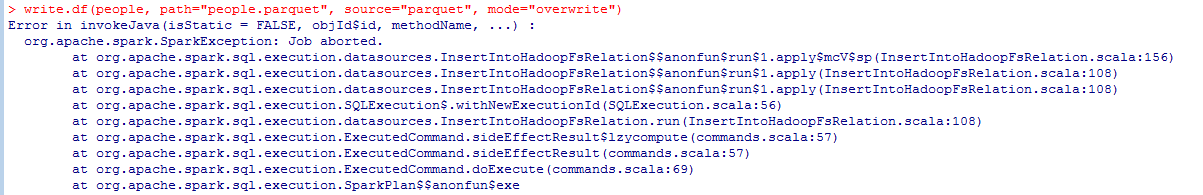


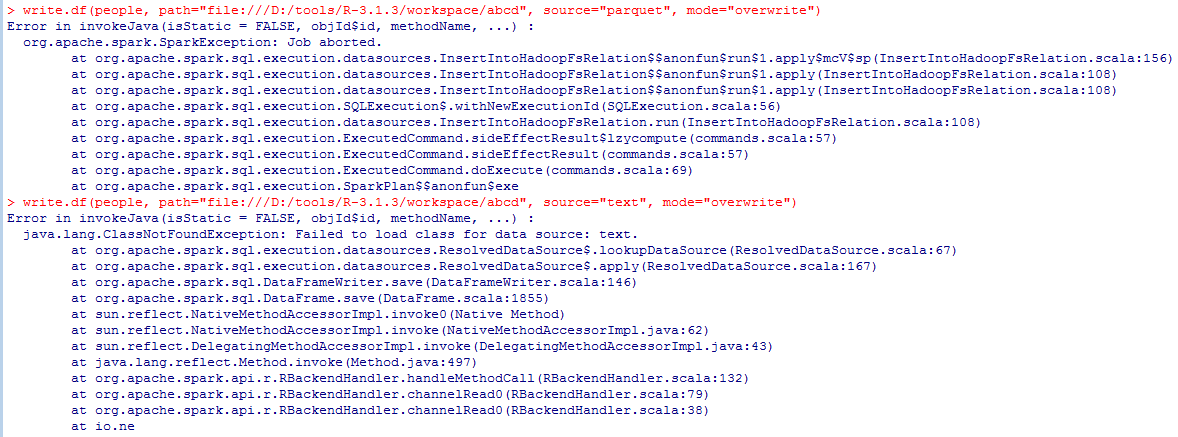
(2)从DataFrames读取数据：





write.df方法测试不成功。报错信息有以下几种：





write.df(people,path=”people.parquet”, source=”parquet”, mode=”overwrite”)

在这里source是数据源文件类型，如果不指定数据源，默认由配置spark.sql.sources.default决定。

mode有四中：append/overwrite/error/ignore

意思分别是：追加/重写/结果文件已有的话打印报错信息/结果文件已有的话不打印报错信息。

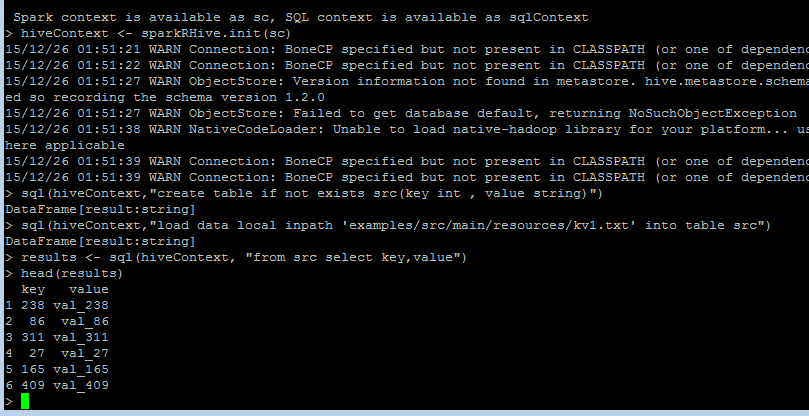
直到Spark1.6默认写mode是override，spark1.7将默认写mode采用error，为了与scala api相一致。

(3)从Hive中读取数据：



此为windows下的报错信息。

在linux下进行此操作：



## 五、利用DataFrames读取其他格式文件。

SparkR通过DataFrames读取数据，默认支持json和parquet文件，读取其他文件需要下载依赖包(<http://spark-packages.org/>)，如下读取csv文件例子：

sparkR --master local[2] --packages com.databricks:spark-csv\_2.10:1.3.0

> st <- read.df(sqlContext,"sample\_tree\_data.csv","com.databricks.spark.csv",header="true")

> head(st)

1 17.99 10.38 122.8 1001 0.1184 0.2776 0.3001 0.1471 0.2419 0.07871

1 1 20.57 17.77 132.9 1326 0.08474 0.07864 0.0869 0.07017 0.1812 0.05667

2 1 19.69 21.25 130 1203 0.1096 0.1599 0.1974 0.1279 0.2069 0.05999

3 1 11.42 20.38 77.58 386.1 0.1425 0.2839 0.2414 0.1052 0.2597 0.09744

或者在进入sparkr shell后重新初始化spark context：

> sc <- sparkR.init(appName="datasource-csv",sparkEnvir=list(spark.executor.memory="1g"),sparkJars="spark-csv\_2.10-1.3.0.jar")

> sqlContext <- sparkRSQL.init(sc)

> st <- read.df(sqlContext,"sample\_tree\_data.csv","com.databricks.spark.csv",header="true")

> head(st)

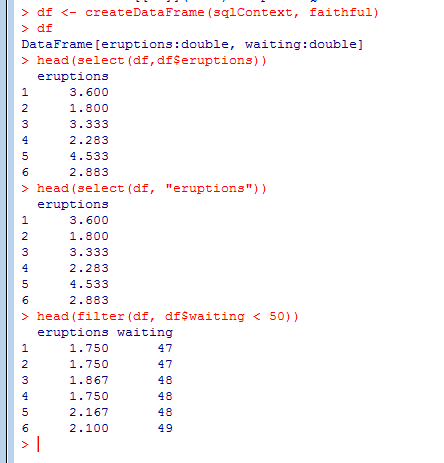
1 17.99 10.38 122.8 1001 0.1184 0.2776 0.3001 0.1471 0.2419 0.07871

1 1 20.57 17.77 132.9 1326 0.08474 0.07864 0.0869 0.07017 0.1812 0.05667

2 1 19.69 21.25 130 1203 0.1096 0.1599 0.1974 0.1279 0.2069 0.05999

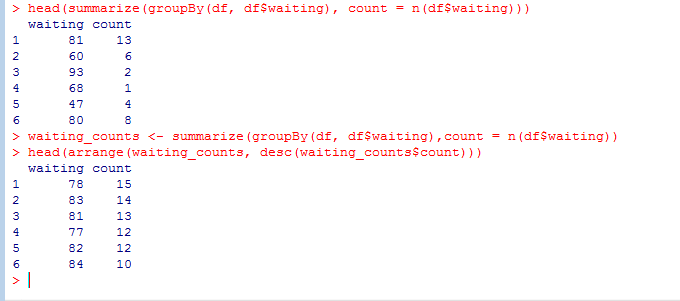
## 六、利用DataFrame进行查询、分组、排序、聚合、过滤、去重等操作。

查询、过滤：



查询所有列，可以使用select(df,”\*”)

分组、聚合和排序：



summarize分类汇总，代码第一行代表先df的waiting列分组，然后对waiting做count操作。

arrange可对DataFrame排序，在这里也可以调用orderBy函数，orderBy函数内部还是调用arrange函数。使用arrange的其他方式：

arrange(df,asc(df$col1), desc(abs(df$col2))) #先对df第一列升序，再对第二列的绝对值做降序

arrange(df,”col1”,decreasing = TRUE) #对col1做降序

arrange(df,”col1”, “col2”,decreasing = c(TRUE,FALSE)) #同样先对df的col1列降序，再对col2升序。

Join操作：

> df1 = read.json(sqlContext,"/vol/lfl/workspace/people1.json")

> df2 = read.json(sqlContext,"/vol/lfl/workspace/people2.json")

> head(df1)

age name

1 NA Michael

2 30 Andy

3 19 Justin

4 20 Mike

> head(df2)

age name

1 30 Andy

2 19 Justin

3 20 Mike

4 30 Jekins

> joinDF <- join(df1,df2,df1$name==df2$name) #默认inner join操作

> head(joinDF)

age name age name

1 30 Andy 30 Andy

2 19 Justin 19 Justin

3 20 Mike 20 Mike

> left\_JoinDF <- join(df1,df2,df1$name==df2$name,"left\_outer")

> head(left\_JoinDF)

age name age name

1 NA Michael NA <NA>

2 30 Andy 30 Andy

3 19 Justin 19 Justin

4 20 Mike 20 Mike

> leftJoinDF <- join(df1,df2,df1$name==df2$name,"leftouter")

> head(leftJoinDF)

age name age name

1 NA Michael NA <NA>

2 30 Andy 30 Andy

3 19 Justin 19 Justin

4 20 Mike 20 Mike

#leftouter和left\_outer的效果是一样的

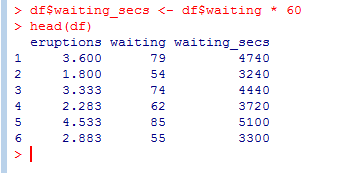
join函数需要四个参数：x，y，joinExpr，joinType

x，y分别为两个DataFrame，joinExpr为列表达式，joinType默认为inner join，有以下几种：

'inner', 'outer', 'full', 'fullouter', leftouter', 'left\_outer', 'left', 'right\_outer', 'rightouter', 'right', and 'leftsemi'

其中leftouter和left\_outer效果相同，rightouter和right\_outer效果相同。

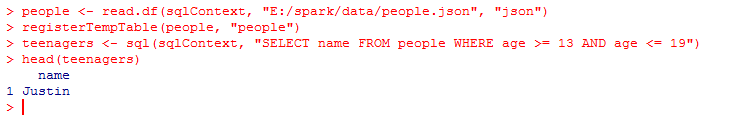
定义新列：



有关DataFrame操作可以到源代码DataFrame.R文件中查看。

有关RRDD操作可以到源代码RDD.R文件中查看。

## 七、利用DataFrames注册临时表，使用sql查询数据。



## 八、改写线上首页抽屉视频点击报表。

homedrawer.R程序代码：

.libPaths(c(file.path(Sys.getenv("SPARK\_HOME"), "R", "lib"), .libPaths()))

library(SparkR)

sc <- sparkR.init(master="local",appName="homedrawer")

sc

#数据源data\_source.txt是经过过滤的ipad的视频点击接口的不到1M的测试数据

contents <- SparkR:::textFile(sc,"/vol/lfl/workspace/data\_source.txt")

#title=黄磊自曝是"小明"真身&ct=综艺&vid=XMTQzMzQ5MTkyMA== y5.home\_shome.channelVideoClick\_96\_4.1\_XMTQzMzQ5MTkyMA==\_1

dsource <- SparkR:::lapply(contents,function(record){

parts <- strsplit(record, "\t")[[1]]

list(pid=parts[1],guid=parts[2],ouid=parts[4],exet\_param=parts[59],refercode=parts[74],osver=parts[11])

})

mm <- SparkR:::filterRDD(dsource,function(record){record[5]!='' })

rdd <- SparkR:::lapply(mm,function(record){

parts <- strsplit(record[[4]],"&")[[1]]

cntemp <- parts[grepl("ct=",parts)]

#频道页

channel\_name <- strsplit(cntemp,"=")[[1]][2]

#视频名称

titletemp <- parts[grepl("title=",parts)]

title <- substr(strsplit(titletemp,"=")[[1]][2],1,5)

refercode <- record[[5]]

#抽屉位置

postemp <- gregexpr('([0-9]+)$',refercode)

if(as.numeric(substr(refercode[1],postemp[[1]],postemp[[1]]+attr(postemp[[1]],'match.length')-1))>10L){

pos <- 11L

}else{

pos <- as.numeric(substr(refercode[1],postemp[[1]],postemp[[1]]+attr(postemp[[1]],'match.length')-1))

}

#vid

vid <- strsplit(refercode,"\_")[[1]][5]

#xuid

guid <- record[[2]]

ouid <- record[[3]]

osver <- record[[6]]

if(osver>'7')

xuid <- ouid

else

xuid <- guid

list(os\_id=52L,device\_type=2L,channel\_name,pos,vid,title,xuid)

})

pvtemp <- SparkR:::lapply(rdd,function(records){

k <- paste(records[[1]],records[[2]],records[[3]],records[[4]],records[[5]],records[[6]],sep="\t")

list(k,1L)

})

#用RRDD操作求pv

pv <- SparkR:::reduceByKey(pvtemp,"+",2L)

#collect(pv)

pv

uvtemp <- SparkR:::lapply(rdd,function(records){

k <- list(os\_id=records[[1]],device\_type=records[[2]],channel=records[[3]],pos=records[[4]],vid=records[[5]],title=records[[6]],xuid=records[[7]])

k

})

#利用DataFrame求uv,方式一

sqlContext <- sparkRSQL.init(sc)

rddDF <- SparkR:::as.DataFrame(sqlContext,uvtemp)

#head(rddDF)

#printSchema(rddDF)

registerTempTable(rddDF,"test\_ipad")

uv1 <- sql(sqlContext,"select os\_id,device\_type,channel,pos,vid,title,count(distinct xuid) as uv from test\_ipad group by os\_id,device\_type,channel,pos,vid,title")

#collect(uv1)

#利用DataFrame求uv,方式二

uv2 <- SparkR:::summarize(groupBy(rddDF,rddDF$os\_id,rddDF$device\_type,rddDF$channel,rddDF$pos,rddDF$vid,rddDF$title),discount = countDistinct(rddDF$xuid))

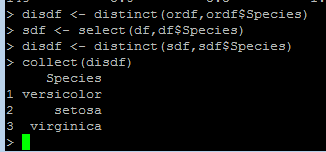
#collect(uv2)

#collect(pv)

## 九、与mllib相结合，目前支持拟合广义线性模型。

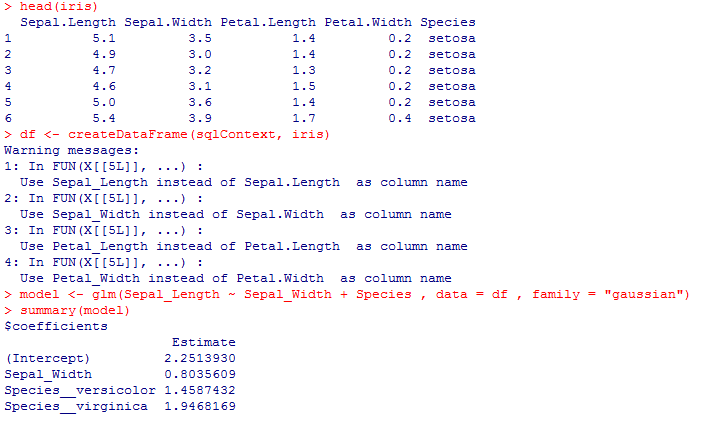
Iris是R当中的鸢尾花数据集，其中包括萼片的长度宽度，花瓣的长度宽度，以及鸢尾花的种类Species构成。

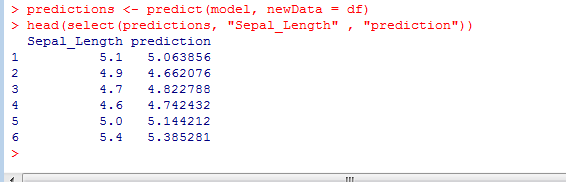
对Species列去重查看鸢尾花种类，使用distinct函数：



Versicolor：云芝 Setosa：清风藤

Virginica：锦葵





glm()出自mllib.R，返回值是PipelineModel类型。主要参数有formula、family、alpha、lambda

formula:拟合模型的描述符号。目前只支持操作符~、+、-、. (点号代表所有)

family”误差分布，线性回归采用高斯分布"gaussian"，逻辑回归用二项式分布”binomial"

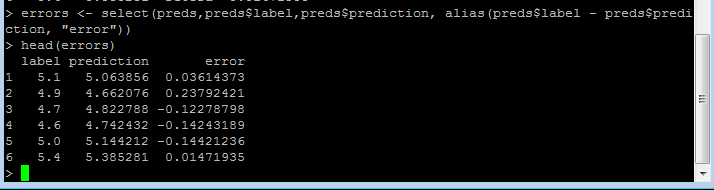
lambda:正则化参数(避免过拟合的惩罚因子)

alpha ：Elastic-net混合参数，其中0≤alpha≤1,elastic-net融合L1和L2范数两种正则化方法，当alpha接近1时elastic-net表现接近L1范数。

具体参见：glmnet's documentation 链接如下：

<https://cran.r-project.org/web/packages/glmnet/index.html>

从预测出的结果中查询结果：



mllib.R目前只有三个函数，glm/predict/summary

summary可以打印出模型的概览数据。类似R语言的summary函数。

## 十、SparkR 1.6新添加内容。

Formula表达式支持”:”。例如int型变量3:6代表3到6，即3,4,5,6

Summary添加最大、最小值、标准差、t值检验等更加完善

解决与R当中函数命名冲突，可以使用SparkR:::函数名或者SparkR::函数名访问函数。

## 十一、优化建议。

R语言在本地创建一个Spark Context对象，其通过JNI调用发起Java Spark Context，

然后由Java Spark Context将其提交到集群由Spark Executor执行相应的R代码。

当执行到各种transform操作时，由于R代码中每一个transform发起一个spark executor执行该操作，所以效率很低。后期R源码当中都采用Pipelined RDD对其进行了优化，是将所有R代码当中的transform汇总到一块，发起一个spark executor进程去执行。

## 十二、遇到的问题。

(1).通过第一种方式安装有些函数可能不支持，例如：

people <- read.df(sqlContext, "E:/spark/data/people.json", "json")

Error: could not find function "read.df"

(2).如果install\_github("amplab-extras/SparkR-pkg",subdir="pkg")方式安装，在进行sparkRSQL初始化时会报错：

sqlContext <- sparkRSQL.init(sc)

Error: 'sparkRSQL.init' is not an exported object from 'namespace:SparkR'

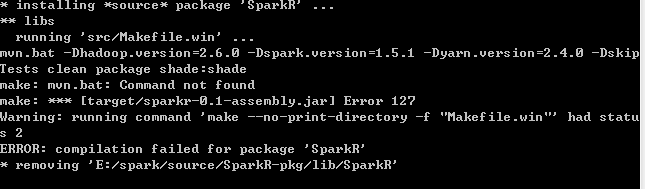
解决方式：

library(devtools)

install\_github("amplab-extras/SparkR-pkg", ref="sparkr-sql", subdir="pkg")

(3).利用install\_github下载安装SparkR时会调用maven并自动进行build，当maven版本是3.3及以上时报错，找不到mvn.bat(注：maven3.3的bin下采用mvn.cmd执行编译，不存在mvn.bat。maven版本3.2及以下bin目录有mvn.bat批处理文件)。

报错信息如下：



解决方式一：选用maven3.2或以下版本

方式二：在maven3.3的bin目录下新建mvn.bat文件，并添加内容: "%~dp0\mvn.cmd" %\*

## 十三、操作过程中经常用的R函数。

查看当前工作目录：

getwd()

修改当前工作目录：

setwd("D:/tools/R-3.1.3/workspace")

查看R存储包路径:

.libPaths()

修改R存储包路径：

.libPaths("D:/tools/R-3.1.3/library")

查看已加载包：

(.packages())

查看某个包中所有函数:

help(package="SparkR")

在当前R terminal中查看某包所有可用函数：

ls("package:SparkR")

浏览器不可用情况下查看某包所有函数：

library(help='graphics')

查看函数(例如summarize函数)信息：

help.search("summarize")或者 ??summarize

联网安装所需包

install.packages('plyr',dependencies = TRUE)

从本地安装包

install.packages("D:/tools/R-3.1.3/rhdfs\_1.0.8.zip")

在当前工作空间不加载该依赖包

detach("package:dplyr")

彻底卸载该某个包

remove.packages("plyr")

加载包：

library("SparkR")

读取输入文件：source(“filename”)

输出文本:sink(“filename”)

输出pdf文件:pdf(“filename.pdf”)

输出jpeg文件:jpeg(“filename.jpeg”)

输出png文件:png(“filename.png”)

输出bmp文件:bmp(“filename.bmp”)