Using The \texttt{iterators} Package

Rich Calaway
doc@revolutionanalytics.com

May 24, 2018

1 Introduction

An \textit{iterator} is a special type of object that generalizes the notion of a looping variable. When passed as an argument to a function that knows what to do with it, the iterator supplies a sequence of values. The iterator also maintains information about its state, in particular its current index. The \texttt{iterators} package includes a number of functions for creating iterators, the simplest of which is \texttt{iter}, which takes virtually any R object and turns it into an iterator object. The simplest function that operates on iterators is the \texttt{nextElem} function, which when given an iterator, returns the next value of the iterator. For example, here we create an iterator object from the sequence 1 to 10, and then use \texttt{nextElem} to iterate through the values:

\begin{verbatim}
> library(iterators)
> i1 <- iter(1:10)
> nextElem(i1)
[1] 1
> nextElem(i1)
[1] 2
\end{verbatim}

You can create iterators from matrices and data frames, using the \texttt{by} argument to specify whether to iterate by row or column:

\begin{verbatim}
> istate <- iter(state.x77, by='row')
> nextElem(istate)
\end{verbatim}
Using The `iterators` Package

```r
Population Income Illiteracy Life Exp Murder HS Grad Frost Area
Alabama 3615 3624 2.1 69.05 15.1 41.3 20 50708

> nextElem(istate)

Population Income Illiteracy Life Exp Murder HS Grad Frost Area
Alaska 365 6315 1.5 69.31 11.3 66.7 152 566432
```

Iterators can also be created from functions, in which case the iterator can be an endless source of values:

```r
> ifun <- iter(function() sample(0:9, 4, replace=TRUE))
> nextElem(ifun)

[1] 3 4 4 1

> nextElem(ifun)

[1] 2 3 5 4
```

For practical applications, iterators can be paired with `foreach` to obtain parallel results quite easily:

```r
> library(foreach)

foreach: simple, scalable parallel programming from Revolution Analytics
Use Revolution R for scalability, fault tolerance and more.
http://www.revolutionanalytics.com

> x <- matrix(rnorm(1e+06), ncol = 10000)
> itx <- iter(x, by = "row")
> foreach(i = itx, .combine = c) %dopar% mean(i)

[1] -0.0069652059 0.0161112989 0.0080068074 -0.0120020610 0.0017168149
[6] 0.0139835943 -0.0078172106 -0.0024762273 -0.0031558268 -0.0072662893
[11] -0.0055142639 0.00155717907 -0.0100842965 -0.0123601527 0.0136420084
[16] -0.0242922105 -0.0126416949 -0.0052951152 0.0216329326 -0.0262476648
[21] 0.0041937609 0.0121253368 -0.0110165729 0.0044267635 0.0080241894
[26] 0.0042995539 -0.0102826632 0.0051185628 -0.0013970812 -0.0172380786
```
2 Some Special Iterators

The notion of an iterator is new to R, but should be familiar to users of languages such as Python. The iterators package includes a number of special functions that generate iterators for some common scenarios. For example, the irnorm function creates an iterator for which each value is drawn from a specified random normal distribution:

```r
> library(iterators)
> itrn <- irnorm(10)
> nextElem(itrn)

[1] 2.03937283 -1.36708005 -0.23984594 -0.78275420 0.46212333 -0.76126267
[7] 0.01966391 -1.42582569 -0.05161063 0.09770317
```

Similarly, the irunif, irbinom, and irpois functions create iterators which drawn their values from uniform, binomial, and Poisson distributions, respectively.

We can then use these functions just as we used irnorm:

```r
> itru <- irunif(10)
> nextElem(itru)
```

Similarly, the irunif, irbinom, and irpois functions create iterators which drawn their values from uniform, binomial, and Poisson distributions, respectively.

We can then use these functions just as we used irnorm:
Using The *iterators* Package

```r
[1] 0.5644920 0.9595622 0.8301415 0.4031848 0.8360185 0.8576464 0.5135703
[8] 0.3587180 0.5551857 0.8740571

> nextElem(itru)

[1] 0.1112257 0.0742968 0.0991537 0.3794068 0.1192921 0.4765969
[7] 0.2839463 0.8236012 0.2800364 0.2775588

The `icount` function returns an iterator that counts starting from one:

```r
> it <- icount(3)
> nextElem(it)

[1] 1

> nextElem(it)

[1] 2

> nextElem(it)

[1] 3
```