Introdução a Linguagem R

Lendo Dados

Delermando Branquinho Filho

Leitura de dados

Existem algumas funções principais de leitura de dados em R. - `read.table`,`read.csv`, para leitura de dados tabulares - `readLines`, para linhas de leitura de um arquivo de texto - `source`, para leitura em arquivos de código R (`inverse` de `dump`) - `dget`, para leitura em arquivos de código R (`inverso` de `dput`) - `load`, para leitura em espaços de trabalho salvos - `unserialize`, para ler objetos R únicos em forma binária

Leitura de arquivos de dados com read.table

A função `read.table` é uma das funções mais utilizadas para a leitura de dados. Ele tem alguns argumentos importantes: - `file`, o nome de um ficheiro ou uma ligação - `header`, lógico indicando se o arquivo tem uma linha de cabeçalho - `sep`, uma string indicando como as colunas estão separadas - `colClasses`, um vetor de caracteres indicando a classe de cada coluna no conjunto de dados - `nrows`, o número de linhas no conjunto de dados - `comment.char`, uma cadeia de caracteres indicando o caractere de comentário - `skip`, o número de linhas a saltar desde o início - `stringsAsFactors`, as variáveis de caracteres devem ser codificadas como fatores?

---

read.table

Para conjuntos de dados de tamanho pequeno a moderado, você geralmente pode chamar read.table sem especificar outros argumentos

```
data <- read.table("foo.txt")
```

R automaticamente - ignora linhas que começam com um # - descobre quantas linhas há (e quanta memória precisa ser alocada) - descobre que tipo de variável está em cada coluna da tabela

Dizer R todas essas coisas diretamente faz R correr mais rápido e mais eficiente. - `read.csv` é idêntico a read.table, exceto que o separador padrão é virgula.

Leitura em conjuntos de dados maiores com read.table

Com conjuntos de dados muito maiores, fazer as seguintes coisas fará sua vida mais fácil e impedirá R de asfixia.

- Leia a página de ajuda para read.table, que contém muitas dicas
- Faça um cálculo aproximado da memória necessária para armazenar seu conjunto de dados. Se o conjunto de dados é maior do que a quantidade de RAM no seu computador, você provavelmente pode parar aqui.
- Defina `comment.char =" "` se não houver linhas comentadas em seu arquivo.

Leitura em conjuntos de dados maiores com read.table

- Use o argumento `colClasses`. Especificar esta opção em vez de usar o padrão pode fazer "read.table" correr muito mais rápido, muitas vezes duas vezes mais rápido. Para usar esta opção, você tem que saber a classe de cada coluna em seu quadro de dados. Se todas as colunas são "numéricas", por exemplo, então você pode apenas definir `colClasses =" numeric "`. Uma maneira rápida e suja de descobrir as classes de cada coluna é a seguinte:

```
initial <- read.table("datatable.txt", nrows = 100)
classes <- sapply(initial, class)
tabAll <- read.table("datatable.txt",
                     colClasses = classes)
```

- Definir `nrows`. Isso não faz R executar mais rápido, mas ajuda com uso de memória. Um overestimate suave é aprovado. Você pode usar a ferramenta Unix `wc` para calcular o número de linhas em um arquivo.

Calculando Requisitos de Memória

Eu tenho um quadro de dados com 1.500.000 linhas e 120 colunas, todos os quais são dados numéricos. Aproximadamente, quanta memória é necessária para armazenar este quadro de dados?

1,500,000 × 120 × 8 bytes/numericos

= 1440000000 bytes

= 1440000000 / 2^{20} bytes/MB

= 1,373.29 MB

= 1.34 GB

.

The Scientist