Integração entre R com Ruby

Na empresa onde eu trabalho, VAGAS Tecnologia, utilizamos Ruby para o desenvolvimento da maioria das aplicações e R para a criação de modelos preditivos. Para que os modelos preditivos sejam utilizados pelas demais aplicações é necessário fazer a integração destes modelos, escritos em R, com o resto das aplicações.

Existem algumas bibliotecas e sistemas que permitem esta integração, por exemplo: RApache[1] , Rserver[2], RinRuby[3] e Rscript[4]. Entre as bibliotecas RApache, Rserver e RinRuby, a biblioteca que parece ser a mais estável, simples de usar e com maior documentação é a RinRuby. O Rscript é um front-end do R para executação de scripts em lote via shell do unix.

Para demonstrar e testar as soluções para integração entre R e Ruby eu criei um modelo usando o algoritmo randomForest do pacote randomForest sobre o conjunto de dados iris. Após criado o modelo, foram testados 4 configurações diferentes (tabelas 1 e 2) da aplicação deste modelo sobre um dataset com diversos tamanhos (i.e., 1,000 ou 10,000 ou 100,000 ou 1,000,000 elementos). Os principais objetivos deste teste foram: (*i*) medir o tempo gasto para processamento de todo dataset, e; (*ii*) identificar eventuais problemas durante a execução da solução, medindo de forma *ad hoc* a robustes de cada solução.

As quatro configurações testadas foram:

1. utilizando apenas o R para executar o modelo - esta configuração serve como *baseline* para os testes, pois não é uma solução que resolve o problema de integração. O script testado é apresentado no código abaixo.

```
library(randomForest)
load("../model/modelIrisRandomForest.rda")

m <- do.call("rbind", sapply(1:1000000, FUN = function(i) c(runif(1,0,1.5),runif(1,0,1.5),runif(1,0,1.5),runif(1,0,1.5)), simplify=
FALSE))
novos <- data.frame(m)
names(novos) <- c('Sepal.Length', 'Sepal.Width', 'Petal.Length' ,'Petal.Width')
pred <- as.character(predict(model, newdata=novos))
```

2. utilizando o RinRuby, onde a parte em Ruby é responsável pela leitura do dataset, o RinRuby envia os comandos para o R e o R executa o modelo sobre o dataset lido, retornando os resultados para a parte em Ruby.
3. utilizando o Rscript. Neste teste foi executado a partir do Rscript o mesmo script que o utilizando no primeiro teste, como apresentado no código abaixo.

```
#!/bin/bash
time Rscript -e "source('test_use_function.R')"
```

[1] http://rapache.net/
[2] http://www.rforge.net/Rserve/doc.html
[3] https://sites.google.com/a/ddahl.org/rinruby-users/
[4] http://stat.ethz.ch/R-manual/R-patched/library/utils/html/Rscript.html

4. utilizando o Rscript e separando a função responsável pela criação do dataset da função responsável pela utilização do modelo com intercâmbio de dados através de arquivo CSV.

Na tabela 1 são apresentados os tempos obtidos nos experimentos 1 (R nativo), 2 (RinRuby) e 3 (Rscript).

Quantidade de exemplos	**R**		**RinRuby**		**Rscript**	
	Média	Desvio Padrão	Média	Desvio Padrão	Média	Desvio Padrão
1.000,000	0,020	0,001	120,148	0,025	0,138	0,002
10.000,000	0,139	0,004	1.208,250	11,440	0,259	0,006
100.000,000	1,487	0,072	---	---	1,960	0,010
1.000.000,000	20,537	0,341	---	---	25,348	0,219

Tabela 1: Tempo gasto para o processamento dos dados (em segundos) nos experimentos com R nativo, RinRuby e Rscript[5]

Como esperado, os tempos medidos no experimento com o R (experimento 1) são melhores que os outros experimentos. O tempo para processamento dos dados utilizando o RinRuby (experimento 2) é bem superior ao tempo utilizado pelo R e Rscript, o que pode inviabilizar a utilização do RinRuby na solução. O Rscript (experimento 3) é quase tão rápido quanto o experimento 1.

Na tabela 2 são apresentados os tempos obtidos no experimento 4 (Rscript com data.frame em formato texto).

Quantidade de exemplos	**Rscript com data.frame em formato texto**					
	Gravação		**Cálculo**		**Total**	
	Média	Desvio Padrão	Média	Desvio Padrão	Média	Desvio Padrão
1.000,000	0,124	0,046	0,133	0,003	0,257	0,046
10.000,000	0,210	0,004	0,232	0,004	0,442	0,008
100.000,000	1,634	0,009	1,926	0,013	3,560	0,020
1.000.000,000	20,928	0,306	31,146	0,298	52,074	0,553

Tabela 2: Tempo gasto para o processamento dos dados (em segundos) no experimento 4

O experimento 4 testa uma configuração mais próxima do real e demonstra que esta abordagem processa os dados gastando menos tempo de processamento do que a solução com RinRuby. No entanto, realizar o intercâmbio de dados via arquivo CSV não parece ser uma solução muito robusta. Ou seja, se a solução a ser desenvolvida não precisa processar um volume grande de dados em um tempo reduzido então o RinRuby parece ser a melhor opção. Caso um dos requisitos da solução seja processar dados num tempo reduzido então é melhor procurar por outra solução, considerando o uso do Rscript.

[5] Não foi possível medir o tempo de processamento para 100.000 e 1.000.000 exemplos usando o RinRuby.