r模型部署实战

周震宇

2018/05/27

平台介绍

- 操作系统: Windows10
- R版本信息：

```
version
```

```
##                _
## platform       x86_64-w64-mingw32
## arch           x86_64
## os             mingw32
## system         x86_64, mingw32
## status
## major          3
## minor          5.0
## year           2018
## month          04
## day            23
## svn rev        74626
## language       R
## version.string R version 3.5.0 (2018-04-23)
## nickname       Joy in Playing
```

今天讲什么

引入模型部署

模型部署的定义；为什么说模型部署在**数据科学**项目中很重要；一般的模型部署的**手段**。

场景实例

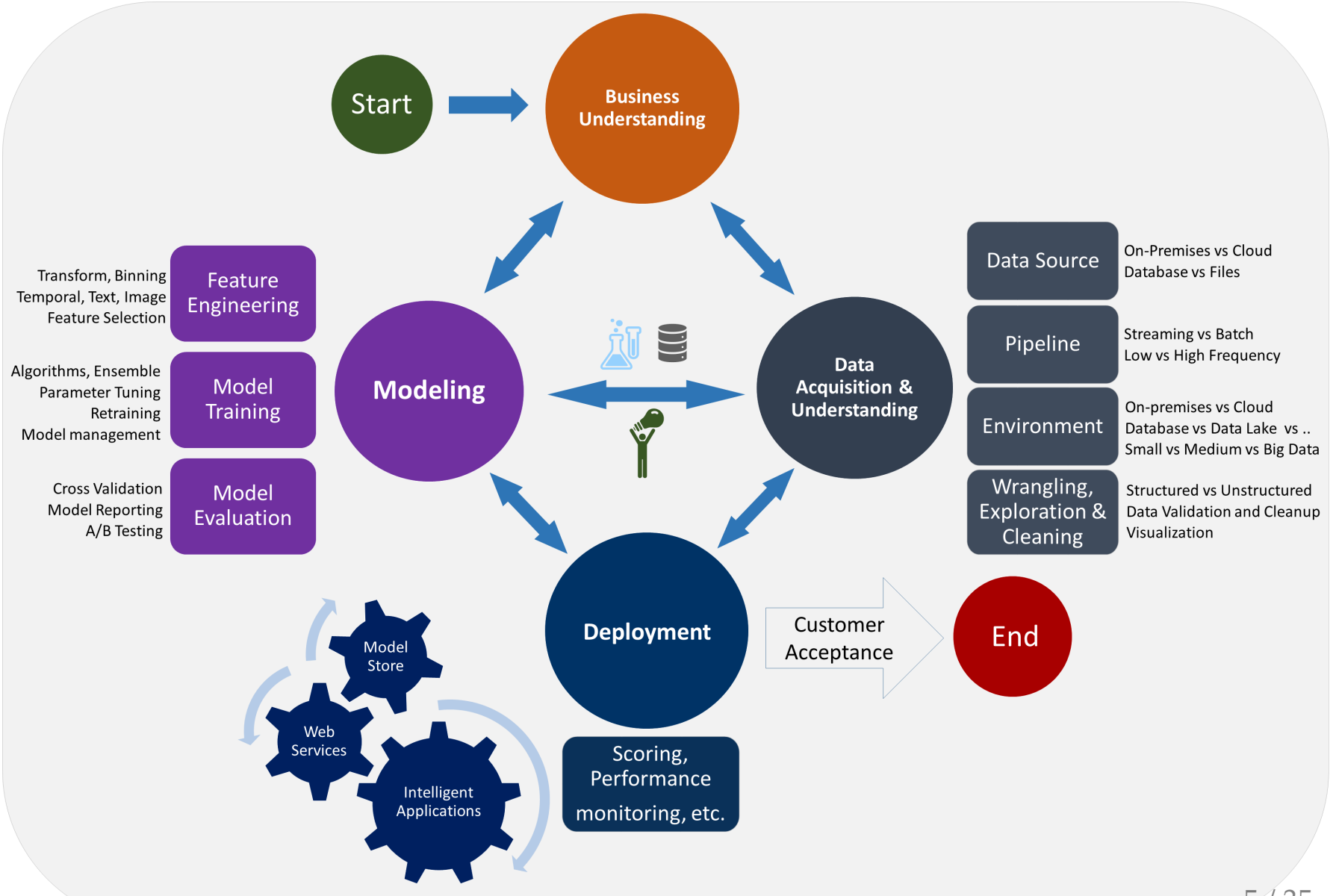
今天用来实践的情景介绍；所用数据与模型。

R包实践

opencpu、fiery、plumbeR的介绍，以及每个包处理实现场景实例的手段。

引入模型部署

Data Science Lifecycle



什么是模型部署

模型部署是将一个调试好的模型运转在生产或者类生产环境中的操作。

它位于数据科学项目生命周期图的终点，是数据分析团队实现一个可用模型的最终一环。模型部署的价值体现在两点：

- 赋予数据科学模型处理实时**流数据**的能力。
- 在实际应用场景中完成模型**效果与性能**的**反馈**，为模型调整提供依据，实现建模开发流程的闭环。

模型部署的手段[1]：

- 在线网站
- 业务线应用程序(Line-of-business applications)
- 移动终端
- 云
- ……

流数据处理方式

依据数据量的大小，实际业务需要等等，处理流数据的基本方式有：

- 实时处理
- 批量处理

本文主要讨论如何在R中借助**云/web服务器**的方式来**部署上线**模型。

[1] 可参考R官网的task view与microsoft文档

统计建模线上与线下的对比

训练一个可用的模型，线上线下环境缺一不可

线下环境

- 离线单机环境
- 实现模型的整个建模流程：特征工程、模型选择、模型评价、参数调节……
- 建模用数据是批量（batch data）的、相对固定的

线上环境

- 在线生产环境
- 数据特点：流数据（streaming data）
- 需要实时反馈

离线单机环境更适合用来做预建模。之后再把单机上调好参数、预测效果良好的模型上线投入实际使用。

技术基础与工具实现

用网络服务这种方式部署模型上线后，需要借助**get&post方法**把特征字段传递给web服务器，服务器将会调用封装好的模型预测并返回参数数值。

浅谈get&post方法

- get方法：从指定的资源请求数据。
- post方法：向指定的资源提交要被处理的数据。

不同的包对用这两种方法实现响应请求的方式略有不同。

web service in R

plumber, fiery, opencpu, httpuv……

场景实例

场景实例：

垃圾邮件拦截

流程：

1. 根据线下数据训练模型
    - 数据来源：ElemStatLearn包spam数据集。
2. 从线上mysql数据库中获取新数据[1]
3. 将数据传递至部署在web服务器上的模型并返回预测值
4. 使用预测值来决策

[1]当然用存储缓存的redis更符合实际场景，这里为简化操作与降低配置难度

R包实践

1. httpuv

> Allows R code to listen for and interact with HTTP and WebSocket clients, so you can serve web traffic directly out of your R process.
>
> This is a low-level library that provides little more than network I/O and implementations of the HTTP and WebSocket protocols.[1]

为plumber、fiery与opencpu等一众用以网络服务的R包提供了处理http请求并搭建网页服务的实现工具。

[1] httpuv官网文档

一个示例

httpuv提供的echo例子

更接近低层，需要编辑html文本

```
library(httpuv)

app <- list(

  ……

    list(
      ……
      body = paste(
        sep = "\r\n",
        "<!DOCTYPE html>",
        "<html>",
        "<head>",
        '<style type="text/css">',
        'body { font-family: Helvetica; }',
        'pre { margin: 0 }',
        '</style>',
        "<script>",
        ……
```

我们试试执行

```
demo('echo',package = 'httpuv')
```

2. opencpu

- R环境中高可用网络服务的前辈
- 2013.09.12 第一版
- 文档示例详尽
- API接口功能完善，便于测试

实现步骤

1. 把要上线的功能写个R包[1]
2. 编译安装R包
3. 开启opencpu的服务器部署模型[2]
4. 连接该端口

[1] 没有R包开发经验的可参考这个

[2] opencpu提供了一个可供测试的页面，需要用浏览器访问http://localhost:5656/ocpu/test

实现步骤

编辑预测函数并保存为PredData.r

```
getdata <- function(id = 1){
  ...
  return(t(as.matrix(z)))
}


xgbpred <- function(id = 1){
  v = xgboost:::predict.xgb.Booster(object = xgbmodel, newdata = getdata(id))
  v = as.character(v)
  return(list(class = v,
              id = id))
}

linearpred <- function(id = 1){
  v = glmnet:::predict.cv.glmnet(object = glmmodel, newx = getdata(id), s = "lambda.min"
  v = as.character(v)
  return(list(class = v,
              id = id))
}
```

实现步骤

创建R包SpamModelOpencpu并编辑文档

```
Package: SpamModelOpencpu
Type: Package
Title: A package used for presentation in China-R report.
Version: 0.1.0
Author: Travis Zhou
Maintainer: Travis Zhou <zhouzy1293@foxmail.com>
Description: A package used for presentation in China-R report.
    Present how to use opencpu to deploy a model.
License: What license is it under?
Encoding: UTF-8
LazyData: false
Imports: xgboost, RMySQL
RoxygenNote: 6.0.1
NeedsCompilation: no
Packaged: 2018-05-19 08:50:22 UTC; Travis
Built: R 3.5.0; ; 2018-05-19 08:50:27 UTC; windows
```

实现步骤

运行单机版的opencpu服务器

```
library(opencpu)
ocpu_start_server()
```

输出：

```
[2018-05-23 19:24:58] OpenCPU single-user server, version 2.0.5
[2018-05-23 19:24:58] Starting 2 new worker(s). Preloading: opencpu, lattice
[2018-05-23 19:24:58] READY to serve at: http://localhost:5656/ocpu
[2018-05-23 19:24:58] Press ESC or CTRL+C to quit!
```

curl一下

```
curl http://localhost:5656/ocpu/library/SpamModelOpencpu/R/xgbpred/json -d 'id=1'
```

返回

```
{
  "class": ["0.875068128108978"],
  "id": [1]
}
```

fiery

八卦

- 2016.07.04 第一版
- shiny：一款温和的工具（do gentle job）。
- fiery：增强了网络服务的功能（reinforced by the name of the package）。
- 同样依赖于httpuv
- 目前使用R包reqres来处理response、request对象

实现步骤

STEP1

首先要开启一个网络服务实例并设置ip地址、端口

```
suppressPackageStartupMessages(library(fiery))
suppressPackageStartupMessages(library(RMySQL))
suppressPackageStartupMessages(library(xgboost))

# 开启一个网络服务实例
app = Fire$new()

# 设置ip 地址和端口号
app$host <- "127.0.0.1"
app$port <- 9123
```

实现步骤

STEP2

启动服务，开启监听，加载模型。

```
# Step1:启动服务，开启监听
app$on("start", function(server, ...) {
  message(sprintf("Running on %s:%s", app$host, app$port))
  model <<- readRDS("model.rds")
  message("Model loaded")

})
```

实现步骤——STEP3

响应http请求，若请求路径为hello或predict，执行对应的函数。

- /hello页面：介绍页面
- /predict页面：输入id，返回预测值的页面。

```
app$on('request', function(server, id, request, ...) {

  response = request$respond()
  ## Step3:获取请求的path，一旦判断为 /predict 则进行预测
  path <- get("url", envir = request)
  message(path)
  ## 索引介绍页面
  if (grepl("hello", path)){
    message('Welcome to the online prediction page')
    response = request$respond()
    response$body = 'Welcome!\n Type the url according to the Email\'s data id \n
    Example: 127.0.0.1:9123/predict?id=235'
    response
  }
  if (grepl("predict", path)) {
    message('reach this procesure.')
    ##Step3.1 获取并解析query string，

    ##query期待处理为 id=##
    query  <- get("querystring", envir = request)
    message(query)
    ## 解析query，大概传递的是类似这个：parseQueryString("?foo=1&bar=b%20a%20r")
    ## input 解析出来是 list 对象
    input <- shiny::parseQueryString(query)
    message(sprintf("Input: %s", input$'?id'))
```

实现步骤

接上一页

```
    ## Step3.2 定义获取数据的函数
    ## 这里模拟从线上mysql提取数据的逻辑，传入待查数据的id
    ## 若id不存在，那么返回报错

    getdata <- function(id = 1){
      con = dbConnect(MySQL(),user = 'test',password = '12345678',dbname = 'spam_mail')
      sqlquery = paste0('SELECT * FROM spam WHERE id = ',id)
      print(sqlquery)
      res <- dbSendQuery(con, sqlquery)
      data <- dbFetch(res, n=-1)
      z <- unlist(data[,-1])
      # 清空con的查询结果并关闭连接
      dbClearResult(dbListResults(con)[[1]])
      dbDisconnect(con)
      return(t(as.matrix(z)))
    }

    ## Step3.3 进入模型预测环节
    ## 声明返回 res 是一个 list，传递参数为 input$id

    res <- list()
    res$v <-  xgboost:::predict.xgb.Booster(object = model, newdata = getdata(input$'?id
    message(res$v)
    response$body <- jsonlite::toJSON(res, auto_unbox = TRUE, pretty = TRUE)
    response$status = 200L
  }
  response
})
```

plumber

包的介绍

- 2016.04.14第一版
- 语法简洁易懂
- 学习曲线平缓
- 响应很快
- 官方文档示例详尽

一个实例：

```
#Example1.r

#' @param event
#' @get /echo

function(event = ''){
  list(msg = paste0(event, " is awesome!"))
}


#' @param spec
#' @get /plot
#' @png

function(spec){
  library(ggplot2)
  myData <- iris
  title <- "All Species"

  # Filter if the s pecies was specified
  if (!missing(spec)){
    title <- paste0("Only the '", spec, "' Species")
    myData %+% subset(iris, Species == spec)
  }

  print(ggplot(myData,aes(Sepal.Length, Petal.Length)) + geom_point() +
        ggtitle(title) + labs(x="Sepal Length", y="Petal Length"))
}
```

接下来打开一个终端并且执行

```
library(plumber)
r <- plumb("Example1.r")  # 需要把路径切至文件'plumber.R'所在位置
r$run(port=2018)
```

这样我们便得到了一个可用的API。

案例实现

endpoint与filter

endpoint

```
#' 对request提取用户id并做预测
#' @get /predict
function(id){
  getdata <- function(id = 1){
    con = dbConnect(MySQL(),user = 'test',password = '12345678',dbname = 'spam_mail')
    sqlquery = paste0('SELECT * FROM spam WHERE id = ',id)
    print(sqlquery)
    res <- dbSendQuery(con, sqlquery)
    data <- dbFetch(res, n=-1)
    z <- unlist(data[,-1])
    # 清空con的查询结果并关闭连接
    dbClearResult(dbListResults(con)[[1]])
    dbDisconnect(con)
    return(t(as.matrix(z)))
  }
  res =  xgboost:::predict.xgb.Booster(object = model, newdata = getdata(id))
  res
}
```

filter

监听请求，返回请求的信息

```
#' 监听请求
#' @filter logger
function(req){
  model <<- readRDS("model.rds")
  cat(as.character(Sys.time()), "-",
      req$REQUEST_METHOD, req$PATH_INFO, "-",
      req$HTTP_USER_AGENT, "@", req$REMOTE_ADDR,"\n")
  plumber::forward()
}
```

每次像服务器发送request都会返回一条记录

```
2018-05-23 20:10:12 - GET /predict - curl/7.55.1 @ 127.0.0.1
2018-05-23 23:03:10 - GET /predict - Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.117 Safari/537.36 @ 127.0.0.1
```

设定输出格式

plumbeR可以依据需求设定输出的格式：如果你想要得到一副图片，那么可以用@jpeg 或 @png[1]

Annotation[2]	Content Type	Description/References
@json	application/json	jsonlite::toJSON()
@html	text/html; charset=utf-8	Passes response through without any additional serialization
@jpeg	image/jpeg	jpeg()
...	...	...

[1] 对于ggplot2的绘图对象，若想要正确打印，需使用print()函数

[2] 来源于plumbeR官方文档输出格式设置

设定输出格式

```
#' 对request提取用户id并做预测
#' @serializer unboxedJSON
#' @get /predict_unboxed
function(id){
  getdata <- function(id = 1){
    ...
  }
  res =  xgboost:::predict.xgb.Booster(object = model, newdata = getdata(id))
  res
}
```

在浏览器中输入http://localhost:2018/predict?id=1 与
http://localhost:2018/predict_unboxed?id=1 会分别得到

```
{"id":["1"],"v":[0.8751]}
{"id":"1","v":0.8751}
```

安全认证

部署在本机的端口上，无需考虑安全认证问题。然而，当把模型部署在一个小型局域网或者开放的网络上面时，就需要对安全认证有所设置。

一般的，并不会允许所有设备均能访问线上模型。

可以用plumbeR的filter编写一个安全认证的小插件

```
function(req, res){
  if (is.null(req$username)){
    res$status <- 401 # Unauthorized
    return(list(error="Authentication required"))
  } else {
    plumber::forward()
  }
}
```

响应测试

企业级应用中，响应时间是衡量取舍一个工具的重要方面。

```
#opencpu响应时间测试
microbenchmark::microbenchmark(system('curl http://localhost:5656/ocpu/library/SpamModel
#fiery响应时间测试
microbenchmark::microbenchmark(system('curl 127.0.0.1:9123/predict?val=1'))
#plumber响应时间测试
microbenchmark::microbenchmark(system('curl 127.0.0.1:2018/predict?id=1'))
```

结果：

```
                                                                    expr      min
system("curl http://localhost:5656/ocpu/library/SpamModelOpencpu/R/xgbpred/json -d \\"i
       lq     mean   median       uq      max neval
1.699469 1.779102 1.808699 1.830126 2.772285   100


                                      expr      min       lq     mean   median       uq
system("curl 127.0.0.1:9123/predict?val=1") 105.5705 206.5052 216.8306 212.6028 219.096

                                     expr     min       lq     mean   median       uq
system("curl 127.0.0.1:2018/predict?id=1") 204.974 207.8494 214.0799 212.2421 216.8809
```

参考资料

1. opencpu官方网站
2. httpuv介绍文档
3. 利用R和opencpu搭建高可用的HTTP服务——by刘思喆
4. plumbeR官方网站
5. 开发 R 程序包之忍者篇——by谢大

谢谢！

本幻灯片由 R 包 **xaringan** 生成；