Deploying Data Mining Results

Web Applications using Shiny

L. Torgo

ltorgo@fc.up.pt

Faculdade de Ciências / LIAAD-INESC TEC, LA
Universidade do Porto

Jun, 2017

Deploying Your Results

- Bringing your results to the users
- Testing on real world environments
- Commercializing your results

Some Potential Difficulties

- Users' lack of knowledge about data mining
- Software requirements for running your results
- Different software environments (e.g. OS, hardware, etc.)
- Maintaining different versions

Some Benefits of Deploying Your Results

- Feedback from real users
- Incorporate feedback - refinement of your solution
- Real world testing

Web Applications

- Applications that runs in a web browser
- Increasingly popular due to:
  - the widespread use of browsers
  - ability to maintain a single version of the software on the server side
  - cross-platform compatibility

Web Applications

- Some Benefits of Web Apps:
  - no complex installation and upgrading processes
  - “no” requirements from the client side (simply a compatible browser)
  - cross-platform compatibility
  - extending to other devices (smartphones, tablets, etc.)
- Some Drawbacks of Web Apps:
  - still some sacrifice to usability
  - eventual connectivity difficulties
  - privacy concerns

R Web Apps using Shiny

- Shiny (http://shiny.rstudio.com) is a web application framework for R
- It has a very intuitive and easy workflow that allows developing web apps very easily
- It allows you to easily deploy your data mining results through a web app
- It is integrated with recent versions of RStudio

NYU STERN
MS in Business Analytics

Installing Shiny

- Shiny can be installed as any R package:

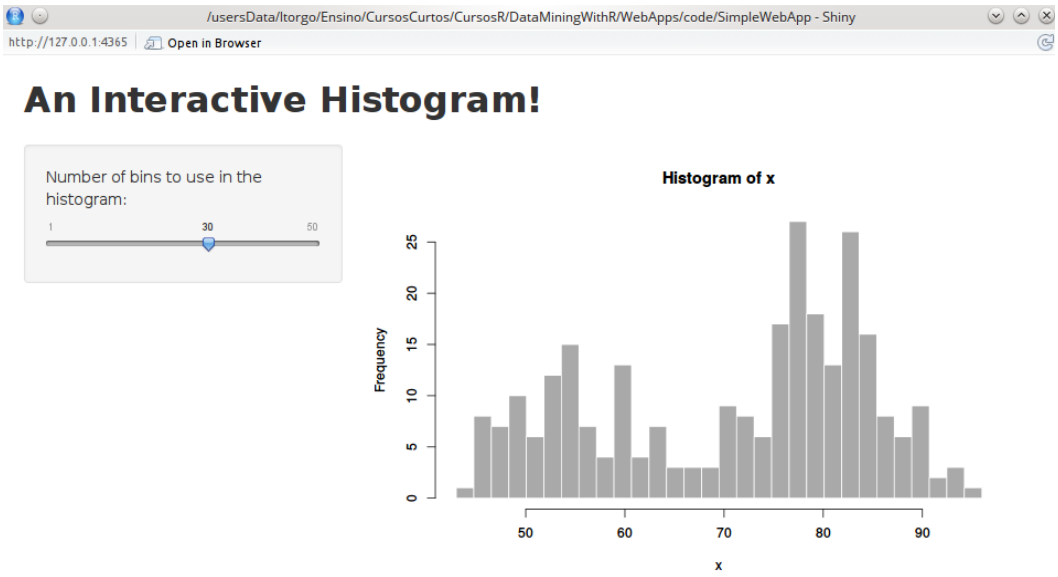
```
install.packages("shiny")
```

- It is strongly integrated with the latest versions of RStudio
- Using RStudio will facilitate your task, but it is not mandatory to have web apps in R

NYU STERN
MS in Business Analytics

A Simple Illustration

An Interactive Hitogram



NYU STERN
MS in Business Analytics

A Simple Illustration

The `ui.R` file

```
library(shiny)
# Define UI for the Web App
shinyUI(fluidPage(

  # Application title
  titlePanel("An Interactive Histogram!"),

  # Sidebar with a slider input for the number of

  # bins
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins",
        "Number of bins to use in the histogram:"
        min = 1,
        max = 50,
        value = 30)
    ),

    # Show a plot of the generated distribution
    mainPanel(
      plotOutput("distPlot")
    )
  )
))
```

The `server.R` file

```
library(shiny)
# Define server logic to draw the histogram
shinyServer(function(input, output) {

  # Expression that generates a histogram. The
  # expression is wrapped in a call to renderPlot
  # to indicate that:
  #  1) It is "reactive" and therefore should be
  #  automatically re-executed when inputs change
  #  2) Its output type is a plot

  output$distPlot <- renderPlot({
    x    <- faithful[, 2]  # Faithful Geyser data
    bins <- seq(min(x), max(x),
                length.out = input$bins + 1)

    # draw the histogram with the specified
    # number of bins
    hist(x, breaks = bins, col = 'darkgray',
         border = 'white')
  })

})
```

NYU STERN
MS in Business Analytics

The Structure of a Shiny Web App

- It should be contained in a folder that gives the name to the App
- The folder should contain at least two files:
  - `ui.R` that controls the layout and appearence of the App
  - `server.R` that defines the instructions that shiny needs to build the App
- **Note**: you may also create a shiny web app on a single file

The Structure of a Shiny Web App

Single file version

- You may put all code in a single file named `app.R`
- The file should be stored in a folder with the name of the web app
  - It may become cumbersome for large apps
  - It does not make so obvious the separation between user interface and the server side code
- Below a simple template for this approach

```
## The content of the "app.R" file
library(shiny)

## The user interface
ui <- fluidPage()

## The server-side code
srv <- function(input, output) { }

## The app
shinyApp(ui = ui, server = srv)
```

Running a Shiny Web App

- Through the function `runApp()`

```
library(shiny)
runApp("AppFolderName")
```

- Using a specific button at RStudio GUI

Taking a Closer Look at "ui.R"

The `ui.R` file

```
shinyUI(fluidPage(

    titlePanel("App Title"),

    sidebarLayout(

        sidebarPanel(),

        mainPanel()

        )
    )
)
```

- `shinyUI()` contains the user interface
- `fluidPage()` is one type of web page, containing a fluid layout consisting of rows and columns. Fluid pages make sure their content scale in realtime to the browser width
- `titlePanel()` creates a title panel in the page
- `sidebarLayout()` allows you to define a layout with a sidebar and a main area
- `sidebarPanel()` defines the content of the side bar
- `mainPanel()` defines the content of the main panel forming the side bar layout

Taking a Closer Look at "server.R"

The `server.R` file

```
shinyServer(
    function(input, output) {
        output$distPlot <- renderPlot({
              bins <- seq(min(x), max(x), length.out = input$bins + 1)
              hist(x, breaks = bins, col = 'darkgray', border = 'white')
            })
    }
)
```

- `shinyServer()` defines the server-side logic of the App. It is a function that is called the first time the browser loads the page. It must take parameters "input" and "output".
- `output` is a list with as many components as there are output elements in the UI. You define here how the output is generated
- `input` is another list with as many components as there are user interface input elements. You can use this list to get the user inputs to the interface.

NYU STERN
MS in Business Analytics

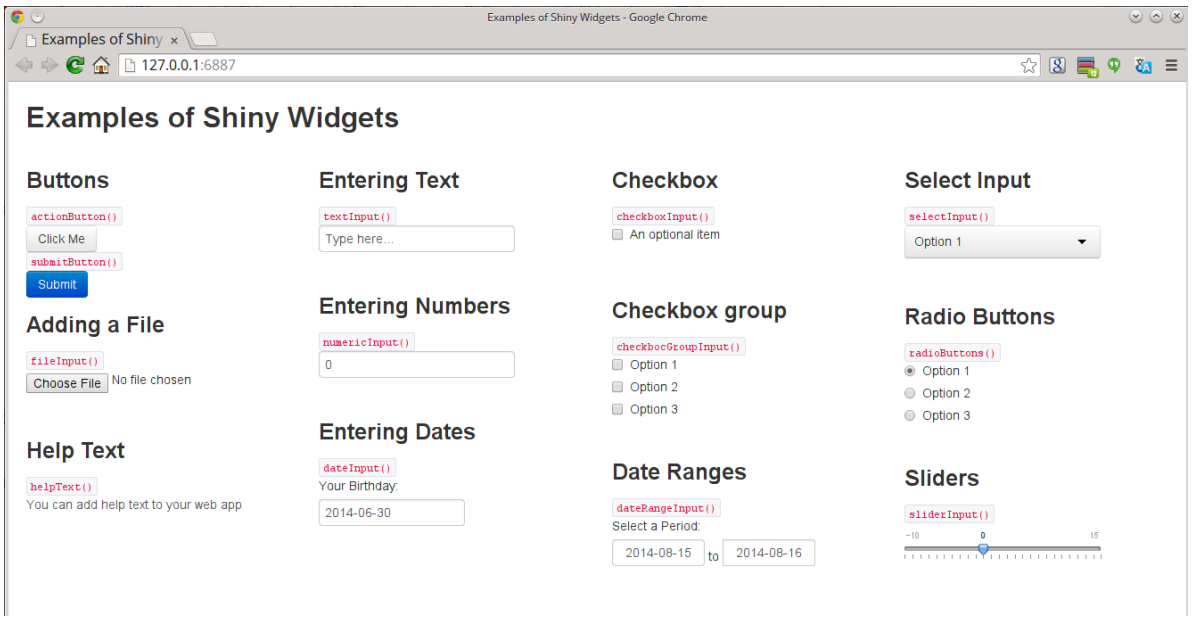
Going Back to the Example

```
# Define UI for the Web App
shinyUI(fluidPage(

  # Application title
  titlePanel("An Interactive Histogram!"),

  # Sidebar with a slider input for the number of

  # bins
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins",
        "Number of bins to use in the histogram:"
        min = 1,
        max = 50,
        value = 30)
    ),

    # Show a plot of the generated distribution
    mainPanel(
      plotOutput("distPlot")
    )
  )
))
```

```
library(shiny)

# Define server logic to draw the histogram
shinyServer(function(input, output) {

  # Expression that generates a histogram. The
  # expression is wrapped in a call to renderPlot
  # to indicate that:
  #  1) It is "reactive" and therefore should be
  #  automatically re-executed when inputs change
  #  2) Its output type is a plot

  output$distPlot <- renderPlot({
    x    <- faithful[, 2]  # Faithful Geyser data
    bins <- seq(min(x), max(x),
                length.out = input$bins + 1)

    # draw the histogram with the specified
    # number of bins
    hist(x, breaks = bins, col = 'darkgray',
         border = 'white')
  })

})
```

NYU STERN
MS in Business Analytics

Some Shiny Widgets



Knowing More About Shiny Widgets

Shiny Widgets Gallery
http://shiny.rstudio.com/gallery/

Output Reacting to User Interaction

Reactive output has to do with content in web apps that is dependent on user interaction with the app (through widgets)

Including Reactive Output

- Having reactive output in a page involves two steps:
  1. Include the object content in the page (in “ui.R”)
  2. Tell shiny how to get the object content (in “server.R”)
- If the object content depends on the value of some shiny widget(s) you have reactive output

Output Functions

- Output functions (`*Output()`) tell Shiny where to display the content of an R object in the page
- Examples:
  - `plotOutput()` can be used to include a plot
  - `tableOutput()` can be used to include the contents of a table
  - `textOutput()` can be used to include text
- There are more functions (all following the naming convention `*Output()`)

NYU STERN
MS in Business Analytics

Building Content

- In “server.R” you tell Shiny how to build the objects that will be placed in the page
- Examples:
  - Non-reactive content

```
shinyServer(function(input, output) {
     output$textField <- renderText({
          paste("The square of 2 is ",2^2)
     })

  }
)
```

  - The above example assumes that somewhere in “ui.R” there is an output element of the form `textOutput(textField)`

NYU STERN
MS in Business Analytics

Building Content (cont.)

- Example with reactive content

```
shinyServer(function(input, output) {
     output$textField <- renderText({
          paste("The square of",input$userNumber,
                "is",input$userNumber^2)
     })

  }
)
```

- Again there should be `textOuput(textField)` in "ui.R", but also a `numericInput()` widget with name "userNumber" that allows the output to be re-active to whatever input the user inserts in this widget

NYU STERN
MS in Business Analytics

Functions for Building Content

- Examples:
  - `renderText()` for character strings
  - `renderPlot()` for R plots
  - `renderPrint()` for any printed output
  - `renderTable()` table-like output (data frames, matrices, etc.)
- There are more functions (all following the naming convention `render*()`)

NYU STERN
MS in Business Analytics

Widget Values

- The components of the `input` list contain the values of the widgets
- The type of value depends on the widget
- For instance a `dateRangeInput()` will produce a vector with two values, whilst a `numericInput()` widget will produce a single value

NYU STERN
MS in Business Analytics

Code Execution

server.R

```
# some code

shinyServer(
   function(input, output) {

      # more code can be here

      output$myPlot <- renderPlot( {

         # yet more code here

      })
   }
)
```

This code is run once when the Web App is launched

Strongly inspired on RStudio Shiny tutorial

NYU STERN
MS in Business Analytics

Code Execution - 2

server.R

```
# some code

shinyServer(

   function(input, output) {

      # more code can be here

      output$myPlot <- renderPlot( {

         # yet more code here

      })
   }

)
```

This code is run once when a new user visits the Web App

NYU STERN
MS in Business Analytics

Code Execution - 3

server.R

```
# some code

shinyServer(

   function(input, output) {

      # more code can be here

      output$myPlot <- renderPlot( {

         # yet more code here

      })
   }

)
```

This code is run every time the user changes a widget that myPlot depends upon

NYU STERN
MS in Business Analytics

Hands on Shiny Web Apps

1. Load the R data set **AirPassengers** and convert it into an `xts` object. Create a web app that allows the user to select the period of time she/he wants to visualize the number of air passengers.
2. Create a web app that allows the user to check the exchange rates between a pre-defined set of currencies. The app should allow the user to: (i) select the currencies; and (ii) select the past time window of rates to visualize. For the second of these requirements the user should be allowed to select among a set of time units (e.g. day, month, year), and also to indicate a number of these units. This means it should be easy to inspect the exchange rates for the last 5 months or 20 days, for instance.

NYU STERN
MS in Business Analytics

How to Share your Web Apps with Others?

- There are two ways (both with pros and cons):
  1. Share the R code of your App (e.g. “ui.R” and “server.R”)
  2. Share the App as the URL of a Web page

NYU STERN
MS in Business Analytics

Sharing the Code

- You send your user the contents of your Web app folder (preferably zipped)
- Your user opens R and provided it has Shiny installed simply runs:

```
library(shiny)
runApp("yourFolderName")
```

- As an alternative you may host your zip file at some web page you have access and the user runs:

```
library(shiny)
# you may try this one!"
runUrl("http://www.dcc.fc.up.pt/~ltorgo/MSBA/SimpleWebApp.zip")
```

- Other possibilities exist through hosting sites like GitHub and Gist and the functions `runGitHub()` and `runGist()`

NYU STERN
MS in Business Analytics

Sharing the Code - pros and cons

- Advantages:
  - Very simple to share - just Zip the folder and you are done
- Disadvantages:
  - Your user needs to have R and the necessary packages installed
  - Your user needs a minimal familiarity with running R and executing commands in R
  - Your app is dependent on the user software environment that you do not control

NYU STERN
MS in Business Analytics

Sharing the App as a Web Page

- Your user will not need to know anything about R!
- RStudio currently provides 3 solutions for this:
  1. ShinyApps.io (https://www.shinyapps.io/)
  2. Shiny Server
  3. Shiny Server Pro

ShinyApps.io

- This is a (free for now) Shiny App hosting service from RStudio
- You create an account and then
  1. Install and then load the **rsconnect** R package
  2. Login into ShinyApps.io and configure the `rsconnect` connection obtaining the authorization information you need to run at your R:

```
setAccountInfo(
    name="ltorgo",
    token="<MYTOKEN>",
    secret="<SECRET>")
```

  3. Move into the folder where your app code is and issue in R:

```
library(rsconnect)
deployApp()
```

  4. Your applications will now be available at a URL related with your account

     `https://ltorgo.shinyapps.io/myWebApp`

Shiny Server

- Shiny Server is a program that builds a web server designed to host Shiny apps.
- It's free, open source, and available from Github.
- Shiny Server runs on Linux servers.

NYU STERN
MS in Business Analytics

Shiny Server

- You need a Linux server running Ubuntu 12.04 or greater (64 bit) or CentOS/RHEL 5 (64 bit).
- If you are not using one of these Linux distributions, you need to compile it from source.

This solution requires access to a Linux web server and technical support to install the program at this server

Installation instructions at
https://github.com/rstudio/shiny-server/blob/master/README.md

NYU STERN
MS in Business Analytics

Shiny Server Pro

- Shiny Server Pro is a paid service to host shiny Apps
- It provides features as:
  - Password authentification
  - SSL support
  - Administrator tools
  - Priority support
  - and more.

More information at
http://www.rstudio.com/products/shiny-server-pro/