FACULTAD DE CIENCIAS AGRARIAS
UNIVERSIDAD NACIONAL DE ROSARIO

GENETICAE

JULIA ANGELINI

TRABAJO FINAL PARA OPTAR AL TITULO DE
ESPECIALISTA EN BIOINFORMÁTICA

DIRECTOR: Gerardo Cervigni
CO-DIRECTOR: Marcos Prunello

AÑO: 2019

Geneticae

Julia Angelini

Licenciada en Estadística – Universidad Nacional de Rosario

Este Trabajo Final es presentado como parte de los requisitos para optar al grado académico de Especialista en _____, de la Universidad Nacional de Rosario y no ha sido previamente presentada para la obtención de otro título en ésta u otra Universidad. El mismo contiene los resultados obtenidos en investigaciones llevadas a cabo en _____, durante el período comprendido entre _____, bajo la dirección de_____.

Nombre y firma del autor

Nombre y firma del Director

Nombre y firma del Co - Director

Defendida: _____ de 20____.

Agradecimientos

En este trabajo final, directa o indirectamente, participaron muchas personas a las que les quiero agradecer.

En primer lugar al Dr. Gerardo Cervigni por confiar en mí y permitirme explorar el mundo de la Bioinformática durante mi tesis doctoral, para que hoy sea parte de mis conocimientos. Al Mgs. Marcos Prunello por acompañarme en el desarrollo del trabajo final, por su dedicación y sus consejos.

Todo esto nunca hubiera sido posible sin el apoyo y el cariño de mis padres, de mi hermano, de Segundo y Kalita. Siempre estuvieron a mi lado, las palabras nunca serán suficientes para agradecerles.

A mis compañeras Jor y Lu, por su ayuda y por compartir excelentes momentos.

A Gaby y Euge mis compañeras de CEFOBI, gracias a ustedes este camino ha sido mas fácil!

A mis amigos, por estar siempre presentes.

Muchas gracias a todos!

Abreviaturas y Símbolos

Resumen

Los programas informáticos se han convertido, hoy en día, en una herramienta básica utilizada por el análisis estadístico como apoyo fundamental a la hora de realizar diferentes operaciones y para facilitar una mayor comodidad a los usuarios. Actualmente, R es uno de los programas más utilizados debido a su potencia y a su distribución como software libre.

Palabras Clave:

Abstract

Keywords:

Índice general

Capítulos **Página**

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

El hombre ha desarrollado el mejoramiento vegetal en forma sistemática desde la aparición de la agricultura y lo ha convertido en un instrumento esencial para la mejora de la producción agrícola en términos de cantidad, calidad y diversidad. Se puede deducir claramente que la sociedad es la beneficiaria del trabajo de fitomejoramiento, debido a que su principal finalidad es obtener mejores resultados de la actividad agrícola, ya sea en alimentos, o en productos que luego serán utilizados en la industria.

Las variedades mejoradas son el resultado del trabajo de desarrollo genético llevado a cabo en los programas de fitomejoramiento, los cuales se extienden a lo largo de varios años y requieren cuantiosas inversiones. La vigencia comercial de las variedades puede extenderse varias décadas, por lo que su elección es crítica para que el productor evite pérdidas económicas por malas campañas y el suministro al mercado sea constante.

La selección de genotipos de buen rendimiento y estabilidad se realiza mediante el análisis de datos provenientes de ensayos multiambientales(EMA), esenciales para analizar la estabilidad del rendimiento. Los EMA comprenden experimentos en múltiples ambientes y son herramientas fundamentales para incrementar la productividad y rentabilidad de los cultivos.

Debido a que las regiones de producción de los principales cultivos cubren áreas ecológicas muy extensas, se observan variaciones en las condiciones climaticas y de suelo, y la aparición de la interacción genotipo ambiente (IGA) es inevitable, provocando respuestas altamente variables. La IGA es considerada casi unánimemente por los fitomejoradores como el principal factor que limita la respuesta a la selección y, en general, la eficiencia de los programas de mejora-

miento. La IGA se puede visualizar al comparar, por ejemplo, el rendimiento de los genotipos en dos ambientes diferentes. Cuando dos genotipos X e Y tienen una respuesta diferencial en ambos ambientes, pero su ordenación permanece sin cambios se dice que la interaccion es no crossover (Figura 1(A)). Sin embargo, la interacción es de tipo crossover cuando hay cambios en el orden de los genotipos (Figura 1(B)). Cuando los genotipos responden de manera similar en ambos ambientes (Figura 1(C)) no hay IGA.

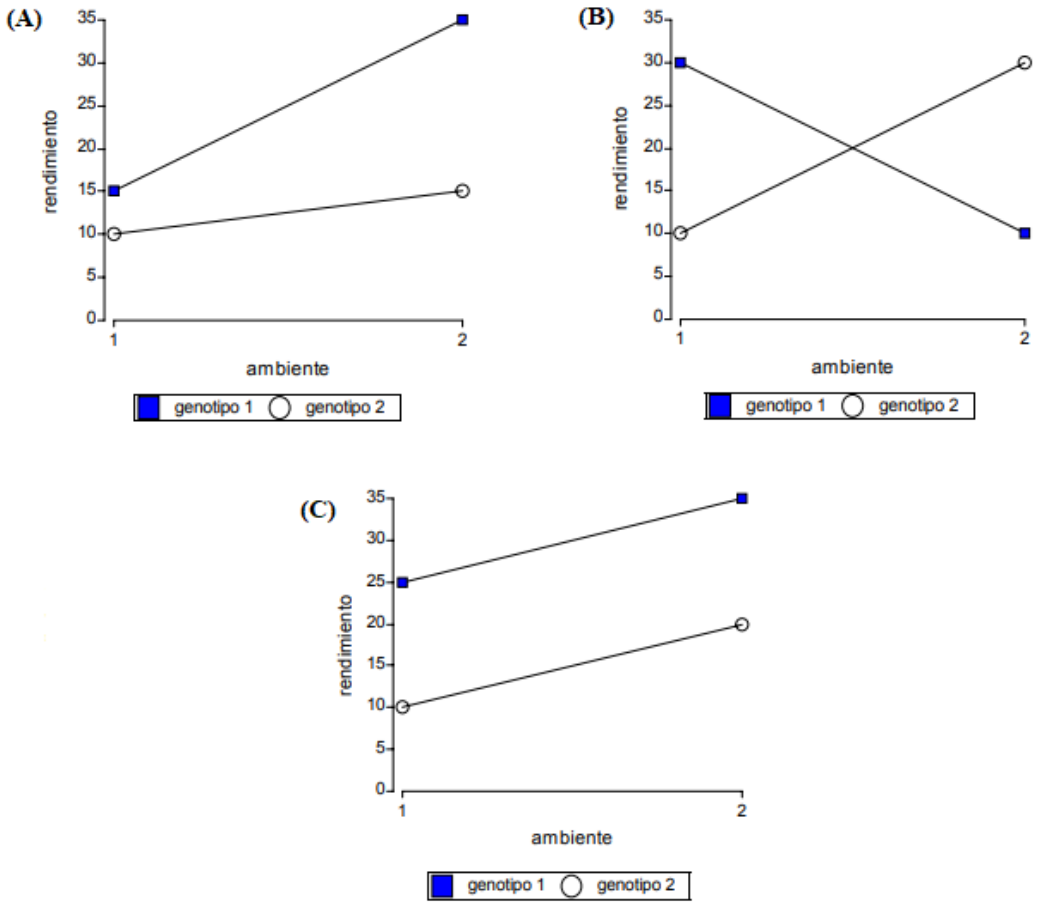


Figura 1.1: Representación gráfica de tipos de IGA: (A)IGA no crossover, (B) IGA crossover y (C) no IGA

Distintos conceptos como regiones ecológicas, ecotipos, mega-ambientes, adaptación específica y estabilidad se pueden analizar a partir de la interacción genotipo-ambiente (Yan y Hunt, 2001).

Comprender la relación entre el rendimiento del cultivo y el medio ambiente ha sido durante mucho tiempo una clave problema para los fitomejoradores y

genetistas. El rendimiento del cultivo, el fenotipo observado, es una función de genotipo (G): variedad o cultivar, medio ambiente (A) y IGA. En general, el rendimiento se puede expresar de la siguiente manera si la interacción entre G y E es significativa:

$$R = G + A + IGA$$

Los investigadores agrícolas han sido conscientes de las diversas implicaciones de IGA en los programas de mejoramiento (Mooers, 1921; Yates y Cochran, 1938). Por ejemplo, la IGA tiene un impacto negativo en la heredabilidad, cuanto menor sea la heredabilidad de un caracter, mayor será la dificultad para mejorar ese caracter mediante la selección. Por lo tanto, información sobre la estructura y la naturaleza de la IGA es particularmente útil para los mejoradores porque puede ayudar a determinar si necesitan desarrollar cultivares para todos los ambientes de interés o si deberían desarrollar cultivares específicos para ambientes específicos (Bridges, 1989). Gauch y Zobel (1996) explicaron la importancia de IGA como: "Si no hubiera interacción, una sola variedad de trigo (*Triticum aestivum* L.) o maíz (*Zea mays* L.) o cualquier otro cultivo rendiria al máximo en todo el mundo, y además la prueba de variedades deberia realizarse en un sólo lugar para proporcionar resultados universales. No habría ruido, los resultados experimentales serían exactos, identificando la mejor variedad sin error, y no habría necesidad de replicación. Entonces, una réplica en un lugar identificaría la mejor variedad de trigo que florece en todo el mundo ".

Un análisis adecuado de la información de los EMA es indispensable para que el programa de mejoramiento de los cultivos sea eficaz. El rendimiento medio en los ambientes es un indicador suficiente del rendimiento genotípico solo en ausencia de IGA(Yan y Kang, 2003). Sin embargo, la aparición de IGA es inevitable y no basta con la comparación de las medias de los genotipos, sino que se debe recurrir a una metodología estadística más aporopiada. Entre los métodos más difundidos para analizar los datos provenientes de EMA se encuentran los modelos de regresión, análisis de variancia y técnicas de análisis multivariado.

Para analizar la IGA se han propuesto varios métodos a través de los años. En 1938 Yates y Cochran trataron de interpretar la interacción calculando la regresión de las medias genotípicas en las medias ambientales, calculadas tomando el promedio de todos los genotipos en ese ambiente. Posteriormente Finlay y Wilkinson (1963) y Eberhart y Russell (1966) retomaron esta práctica, pero con enfoques diferentes.

Shukla (1972) y Cruz Medina (1992) proponen un método práctico para estimar la componente de la interacción GA correspondiente a cada genotipo, incluyendo la posibilidad de efectuar test de hipótesis y permitiendo una posterior extensión del modelo en la interpretación de la inestabilidad. Para ello se considera alguna característica ambiental medible como covariable. Este método también se puede aplicar a los ambientes.

En los últimos años, dos modelos multiplicativos, el modelo de los efectos principales aditivos y interacción multiplicativa (AMMI, *Additive Main effects and Multiplicative Interaction*) y el de regresión por sitio (SREG), han aumentado su popularidad entre los fitomejoradores como una herramienta de análisis gráfico. Estos modelos combinan el análisis de varianza (ANOVA) y la descomposición de valores singulares (SVD) o el análisis de componentes principales (PCA) en la matriz residual de ANOVA. En SREG, el ANOVA se realiza sobre el efecto principal de A mientras que en AMMI se considera el efecto de G y A.

En este contexto, el análisis de datos provenientes de ensayos multiambientales requiere metodología estadística sofisticada cuyas rutinas informáticas se encuentran disponibles en programas desarrollados por diferentes empresas. Esto genera el inconveniente de tener que disponer de todos los programas necesarios para los distintos análisis, atender los requerimientos de formatos de datos usados por cada uno, y comprender los diversos tipos de salidas en las que se ofrecen los resultados obtenidos. Además, algunos procedimientos, especialmente aquellas metodologías recientes, no se encuentran dispobibles, y los costos de las licencias de dichos programas resultan muy elevados. De aqui surge la necesidad de disponer de algún software libre que contemplara la mayoría de las rutinas necesarias para analizar los datos provenientes de EMA. Este problema, puede ser resulto a partir del software R el cual es un lenguaje de programación interpretado. Se trata de un proyecto de software libre distribuido bajo los términos de la *General Public Licence* (GNU). Es el resultado de la implementación del lenguaje S, uno de los lenguajes más utilizados en investigación por la comunidad estadística. A diferencia de otros programas de estadística usuales en el campo de la investigación, R no dispone de una interfaz gráfica con la cual se pueda realizar cualquier análisis, lo cual genera que muchos científicos no lo utilicen. Sin embargo, al tratarse de un lenguaje de programación, las posibilidades que ofrece en cuanto a la manipulación y análisis de los datos es mucho más vasta, ya que los usuarios pueden definir sus propias funciones, personalizar el tipo de

análisis que quieren correr y, a largo plazo, hacer más rápido y eficiente este tipo de tarea.

Una de las características de R es que promueve el hecho de que si alguien crea algo que considera útil para la comunidad pueda ponerlo al alcance de los demas usuarios. Muchas veces, una función creada por algun usuario no resulta sencilla de reutilizar, por lo que se ha introducido la posibilidad de crear paquetes o librerías. Una librería o paquete (*package*) es una colección de objetos creados y organizados siguiendo un protocolo fijo que garantiza un soporte mínimo para el usuario así como la ausencia de errores (de sintaxis) en la programación. Al abrir R se cargan automáticamente una serie de paquetes básicos, como por ejemplo, *base* o *stats*, que contiene las funciones *lm* o *glm*. Sin embargo, R tiene paquetes (unos cuantos miles en la actualidad) .

Al ser un software libre los usuarios tienen la libertad de ejecutar, copiar, distribuir, estudiar, modificar y mejorar el software. Más precisamente, software libre significa que los usuarios de un programa tienen las cuatro libertades esenciales:

1. Ejecutar el programa como lo desee, con cualquier propósito.

2. Estudiar el funcionamiento del programa y modificarlo de modo que realice las tareas que desee. El acceso al código fuente es un prerrequisito para esto.

3. Redistribuir copias para ayudar a los demás.

4. Distribuir copias de sus versiones modificadas a otras personas. Al hacerlo da a toda la comunidad la oportunidad de beneficiarse de sus cambios. El acceso al código fuente es un prerrequisito para esto.

La comunidad de usuarios que programan en R ha ido creciendo notablemente en los últimos años y, al ser un software libre, han ido proporcionando librerías que extienden sus funciones básicas y pueden resultar realmente valiosas (casi indispensables). Entre ellas se encuentran, *plyr*, *lubridate*, *reshape2* y *stringr* para la manipulación de los datos; *ggplot2* y *rgl* para la visualización; *knitr* y *xtable* para la presentación de resultados; entre otros. La lista completa de los paquetes oficiales puede consultarse en CRAN[1]. Además de los oficiales, existen paquetes

[1]CRAN (Comprehensive R Archve Network) es el repositorio oficial de paquetes de R, el lugar donde se publican las nuevas versiones del programa, etc. Contiene la lista completa de paquetes oficiales. https://cran.r-project.org/web/packages/available_packages_by_name.html

que pueden instalarse desde repositorios como, por ejemplo, Github. Sin embargo, no es sencillo encontrar un paquete que puede ser útil para un determinado fin sino que se debe recurrir a varios de ellos para cumplir un determinado objetivo.

Frecuentemente, los mejoradores utilizan una interfaz gráfica para realizar los análisis estadísticos deseados y no tienen un manejo fluido de un lenguaje de programación. En el año 2012 se creó el paquete *Shiny* de RStudio, el cual permite desarrollar aplicaciones Web. En este sentido el paquete Shiny surge como una útil y versátil herramienta que permite acercar la potencia de R a todo tipo de usuarios.

El objetivo del presente trabajo fue: (i) crear un paquete de R que incluya las funciones que permitan analizar la interacción genotipo por ambiente, incluyendo además metodología recientemente publicada que no se encuentra disponible en R; (ii) crear una interfaz gráfica, entre R y el usuario, mediante Shiny que permita realizar los análisis disponibles en el paquete creado.

Capítulo 2

Objetivos

2.1. Objetivo general

Construir un paquete para el programa R, con funciones estadísticas que permitan analizar datos provenientes de EMA, tanto para Windows como Linux, y que cumpla con los estándares de calidad de software. Por otro lado, desarrollar una Shiny APP que permita a los usuarios utilizar las funciones del paquete sin tener que utilizar un lenguaje de programación.

2.2. Objetivos específicos

1. Mostrar el procedimiento para la construcción del paquete de R.
2. Modificar funciones existentes para el análisis de datos provenientes de EMA de manera que sean más flexibles que las actuales.
3. Incorporar metodología recientemente publicada en el paquete de R.
4. Desarrollar una shiny APP para analizar los datos provenientes de EMA.

Capítulo 3

Métodos

3.1. Paquete de R

Una librería o paquete (*package*) es una colección de objetos creados y organizados siguiendo un protocolo fijo que garantiza un soporte mínimo para el usuario así como la ausencia de errores (de sintaxis) en la programación.

3.2. Creación del paquete de R

Los pasos necesarios para la creación de un paquete son:

- Creación de los objetos que contendrá el paquete (funciones y/o datos).
- Creación del esqueleto del paquete.
- Redacción de la documentación.
- Compilación del paquete en Linux y creación de la versión para Windows.
- Instalación.
- Prueba y publicación.

3.2.1. Objetos del paquete

Un paquete puede contener cualquier tipo de objetos de R : funciones, datos etc. Lo primero que debe hacerse es programar las funciones y preparar los datos. El proceso de creación vigila que no hayan errores sintácticos pero no controla si hay errores lógicos.

3.2.2. Esqueleto y estructura del paquete

R proporciona una función *package.skeleton* que permite automatizar el proceso de creación de un paquete creando los directorios, los archivos de documentación y otros objetos necesarios. La siguiente instrucción construye la estructura de un paquete llamado *geneticae*,

```
package.skeleton(name = geneticae)
```

creando una carpeta de nombre *geneticae* con 3 sub-carpetas en el directorio de trabajo y tres archivos sin extensión. Estos ultimos son los siguintes:

- DESCRIPTION: contenido básico para documentar según la descripción del paquete:

  Package: geneticae
  Type: Package
  Title: What the package does (short line)
  Version: 1.0
  Date: 2019-09-21
  Author: Who wrote it
  Maintainer: Who to complain to ¡yourfault@somewhere.net¿
  Description: More about what it does (maybe more than one line)
  License: What license is it under?

- NAMESPACE: R usa un sistema de gestión de espacio de nombres que permite al autor del paquete especificar:

  - las variables del paquete que se exportan (y son, por tanto, accesibles a los usuarios)
  - las variables que se importan de otros paquetes.
  - las clases y métodos S3 y S4 que deben registrarse.

  Este mecanismo queda definido en el contenido del fichero NAMESPACE.

- Read-and-delete-me: contiene algunas instrucciones importantes sobre cómo personalizar el paquete.

Las siguientes son las 3 sub-carpetas creadas:

- La carpeta **data** contiene todos los archivos correspondientes a los datos comprimidos con el nombre con el que fueron creados, con la extensión .rda. Estos no pueden ser modificados.

- La carpeta **man** contiene todos los archivos de extensión .Rd y un archivo por objeto creado (datos o programa). Estos documentos son parte del sistema de ayuda del paquete en PDF y en HTML; por este motivo, la escritura sigue las reglas de LaTeX.

- La carpeta **R** contiene todos los programas fuente, siendo .R la extensión de los mismos.

La documentación es uno de los aspectos mas importantes del código, sin ella, los usuarios no sabrán cómo usar el paquete. R proporciona una forma estándar de documentar paquetes: escribir archivos .Rd en la carpeta man, los cuales utilizan una sintaxis personalizada, basada en LaTeX. Sin embargo, el paquete *roxygen2*, utilizado en este trabajo, permite obtener la documentación de una manera sencilla, proporcionando una serie de ventajas sobre la escritura los archivos .Rd:

- El código y la documentación son adyacentes, de modo que cuando el código se modifique, será fácil actualizar la documentación.

- Inspecciona dinámicamente los objetos que está documentando, para que pueda agregar automáticamente los datos que de otra forma se deben escribir a mano.

- Resume las diferencias en la documentación de los métodos S3 y S4, los genéricos y las clases, por lo que necesita aprender menos detalles.

Además de generar archivos .Rd, *roxygen2* también creará un archivo NAMESPACE y administrará el campo *Imports* del archivo DESCRIPTION.

3.2.3. Compilación e instalación

Una vez creada la documentación se debe chequear el paquete y generar los instaladores con su corresponiente manual. Para ello se utilizan las siguiente funciones:

- *R CMD check* verificará que no haya errores de sintaxis o no se generen warnings. Está compuesto por más de 50 chequeos individuales entre los cuales se encuentran: la estructura del paquete, el archivo descripción, namespace, el código de R, los datos, la documentación, entre otros.

- *R CMD build* compilará el paquete generando un archivo geneticae.tar.gz listo para su instalación en Linux y *RCMD build -binary* generará el archivo para la instalación en Windows.

- *RCMD Rd2dvi –pdf* preparará el manual y *R CMD INSTALL* instalará el paquete dejándolo listo para su uso

3.2.4. Publicación

Un repositorio es el lugar dónde están alojados los paquetes y desde el cuál se pueden descargarlos. Entre los repositorios más populares de paquetes R se encuentran:

- **CRAN**: es el principal repositorio de paquetes de R, está coordinado por la fundación R. Previa a la publicación en este repositorio el paquete debe pasar por diferentes pruebas para asegurar que cumple con las políticas de CRAN.

- **Bioconductor**: se trata de un repositorio específico para bioinformática. Del mismo modo que CRAN, tiene sus propias políticas de publicaciones y procesos de revisión.

- **GitHub**: a pesar que no es específico para R, github es con toda seguridad el repositorio más popular para la publicación de proyectos *open source* (del inglés, código abierto). Su popularidad procede del espacio ilimitado que proporciona para el alojamiento de proyectos *open source*, la integración con git (un software de control de versiones) y, la facilidad de compartir y colaborar con otras personas. Una de sus desventajas es que no proporciona procesos de control.

- **R-Forge** y **RForge**: son entornos de desarrollo de paquetes y repositorios. Eso significa que incluyen control de fuente, seguimiento de errores y otras características. Puede obtener versiones de desarrollo de paquetes de estos.

El paquete *geneticae* se encuentra en GitHub, para instalar el mismo (o cualquiera que se encuentre en dicho repositorio) se deben seguir las siguientes instrucciones:

```
install.packages(remotes)
library(remotes)
install_github(jangelini/geneticae)
```

FALTAN LAS COMILLITAS EN INSTALL Y EN EL USUARIO DE GITHUB.. ME DA ERROR CUANDO LAS PONGO
(Ideas de: TRABAJO FINAL P SHINY)

3.3. Aplicación Web

Una aplicación web es una aplicación o herramienta informática accesible desde cualquier navegador, bien sea a través de internet (lo habitual) o bien a través de una red local. Estas aplicaciones son muy populares hoy en día para los usuarios no expertos, debido a la facilidad de su uso, ya que no es preciso instalar nada en el ordenador, simplemente se accede a través de un navegador. Además se puede acceder desde cualquier dispositivo con conexión a internet, ya sea un ordenador, un smartphone o una tablet, es decir que es independiente del sistema operativo del usuario. Otra gran ventaja es el bajo consumo de recursos, ya que la mayor parte del tiempo estos se consumen en el servidor donde se encuentra alojada la aplicación, que generalmente tiene mucha más potencia de cómputo que cualquier ordenador personal.

3.4. Shiny APP

Shiny es un paquete, que se encuentra instalado por defecto con Rstudio, con el que se pueden desarrollar aplicaciones web interactivas, directamente desde Rstudio sin necesitar conocimientos de HTML, CSS o Javascript. Shiny implementa la programación reactiva (cita de archivo aplicacion shiny) en donde los objetos (gráficos y tablas) que forman la aplicación responden a los inputs de los usuarios, dotando a estos de una gran capacidad de control.

En general, en estas aplicaciones, se distinguen tres pasos en el funcionamiento de la aplicación:

1. El usuario modifica todos aquellos widgets que quedan a su disposición en el navegador (los llamaremos inputs).

2. Los valores de los inputs se envían a R que realiza los análisis indicados.

3. Los resultados de estos cálculos se muestran en el navegador (los llamaremos outputs).

El esquema interno de la aplicación puede observarse en la Figura 3.1.

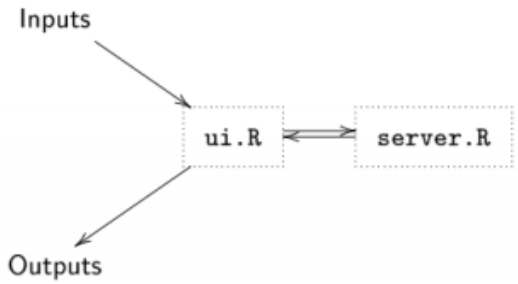


Figura 3.1: Esquema interno de la aplicación.

En las páginas y aplicaciones web se ha estandarizado el uso de ciertos lenguajes, tales como HTML, CSS, PHP o Javascript entre otros. Por defecto Shiny utiliza una plantilla básica de Twitter Bootstrap [18] para crear la interfaz de usuario, pero podemos descargar otras plantillas o crear una propia para personalizar nuestra aplicación. Twitter Bootstrap es un entorno de trabajo desarrollado por empleados de Twitter para fomentar la consistencia entre las herramientas internas, de forma que todas siguieran el mismo estilo. En 2011 Twitter liberó Bootstrap como código abierto, permitiendo que cualquiera lo usara para diseñar sus sitios o aplicaciones web. Contiene elementos de diseño basado en HTML, CSS y Javascript. Una de las mayores ventajas de Bootstrap es que permite crear interfaces web con CSS y JavaScript que adaptan la interfaz dependiendo del tamaño del dispositivo en el que se visualice de forma nativa, es decir, automáticamente se adapta al tamaño de un ordenador o de una tablet sin que el usuario tenga que hacer nada. Esto se denomina diseño adaptable o Responsive Design.

En lugar de usar un CSS propio para la interfaz de la aplicación se ha usado el paquete de R shinythemes [8], publicado por los creadores del propio Shiny. Shinytemes ofrece una serie de estilos básicos para aplicaciones Shiny.

3.5. Creación de la Shiny APP

La instalación de este paquete puede realizarse a través de los menús de Rstudio o simplemente con la siguiente orden:

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

Para crear una Shiny app en *File* se elige la opción Shiny Web App (figura 3.2).

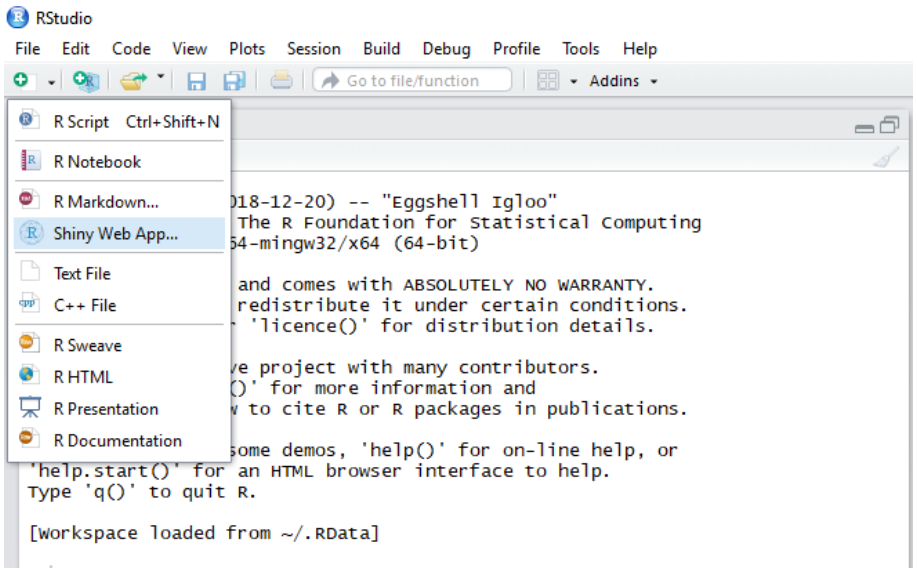


Figura 3.2: Creación de Shiny Web App desde Rstudio

Las aplicaciones Shiny están compuestas por un archivo app.R o dos archivos ui.R y server.R, es decir, se puede partir de un solo fichero que aglutine todo el código o se puede partir de dos archivos que separan la parte cliente de la parte servidora.

En este trabajo, se crea la aplicación Shiny mediante un unico script llamado app.R. El mismo se encuentra en un directorio (por ejemplo newdir/) y la aplicación se puede ejecutar con runApp("newdir"). El script app.R esta formado por

tres componentes:

- ui (*user interfaz*): la interfaz de usuario controla el diseño de la aplicación, recibe los inputs y muestra los outputs en el navegador.
- server, funciones de R que contienen las instrucciones que se necesitan para construir los resultados de los análisis incluidos en la aplicación.
- shinyApp, función que crea objetos de aplicación Shiny a partir de ui / servidor.

El archivo app.R deberá comenzar cargando el paquete Shiny y finalizar con una llamada a shinyApp:

```
library(shiny)
ui<- ...
server<- ...
shinyApp(ui = ui, server = server)
```

La sesión de R estará monitoreando la aplicación y ejecutando las reacciones de la aplicación mientras la aplicación Shiny esté activa, por lo que no podrá ejecutar ningún comando.

La Figura 3.3 muestra el diseño utilizado en la aplicación. Se cuenta con un titulo y diferentes pestañas que conducen a diferentes páginas de la aplicación (**A**). Se cuenta con un panel de barra lateral (**B**), que contiene principalmente widgets con los cuales el usuario puede determinar el análisis que desea realizar y un panel principal (**C**) en el cual se obtenene los resultados del análisis solicitado.

3.5.1. Tipos de componentes de Shiny Web App

Los tipos de componentes de una aplación Shiny son:

- Componentes de Inputs y Outputs. Entre los inputs se encuentran numericInput, sliderInput, textInput, checkboxInput, entre otros. Posibles outputs se encuentran en la tabla 3.1, con las correspondientes funciones para server.R y ui.R
- Componentes de Diseño. Algunos ejemplos se muestra en la tabla 3.2.
- Componentes HTML (tags).

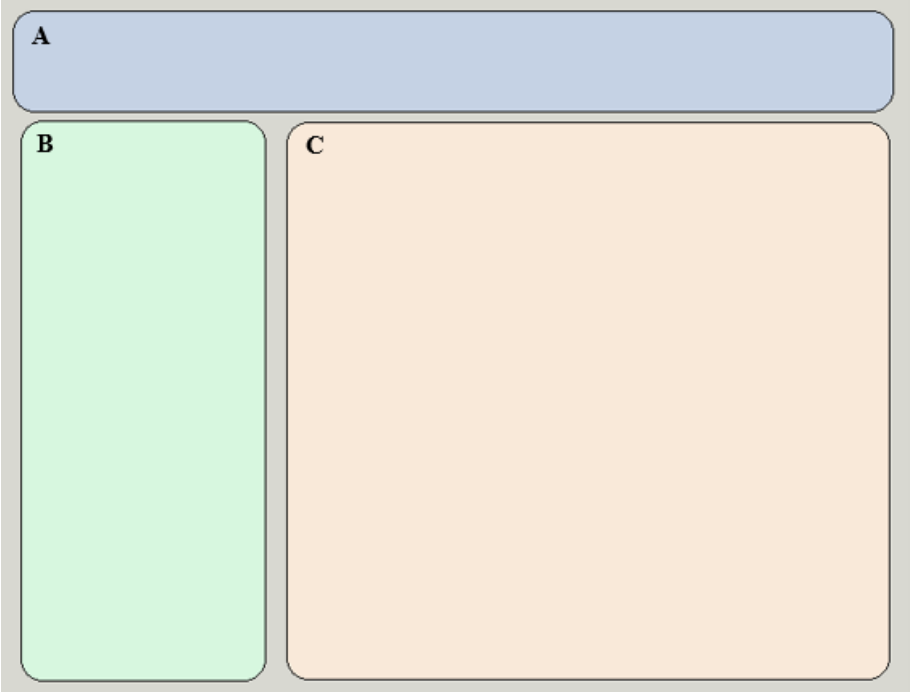


Figura 3.3: Diseño de la aplicación Shiny. **A**: título y pestañas; **B**: área de entrada; **C**: área de resultados.

Componentes de Diseño

- navbarPage(): Crea una página con una barra de navegación de nivel superior.
- tabPanel(): Crea un panel de pestañas.
- sidebarLayout(): Diseña de una barra lateral y el área principal.
- sidebarPanel(): Crea un panel de barra lateral.
- mainPanel(): Crea un panel principal.
- navlistPanel(): Crea un panel de lista de navegación.
- prettyRadioButtons(): Crea botones que permiten seleccionar un elemento de una lista.
- materialSwitch(): Crea un interruptor de palanca para activar o desactivar una selección.

Tabla 3.1: Funciones para Outputs tanto para server.R como para ui.R

server.R	ui.R	espera	crea
renderPlot	plotOutput	Gráfica	Gráfica
renderPrint	verbatimTextOutput, htmlOutput	salida impresa	texto
renderTable	tableOutput	objetos como tablas	tabla simple
renderDataTable	dataTableOutput	objetos como tablas	tabla DataTables.js
downloadHandler	downloadButton, downloadLink		

- pickerInput(): Crea un control de selección de entrada.
- navbarMenu():

Tabla 3.2: Componentes de Diseño de Shiny Web App

Componente	Subcomponente
navbarPage():	tabPanel(), navbarMenu()
navbarMenu()	tabPanel()
navlistPanel()	tabPanel()
titlePanel()	
sidebarLayout()	sidebarPanel() mainPanel() (obligatorio)
sidebarPanel()	
mainPanel()	
tabsetPanel()	
tabPanel()	

Para más información sobre estas componentes ver hoja de referencia de Shiny (Apéndice A).

3.5.2. Compartiendo una Shiny Web App

Una vez creada la aplicación, resulta conveniente ponerlas a disposición de los usuarios. En este caso la Shiny Web App encuentra disponible en el servidor de CONICET `www.cefobi.com`. Además el proyecto se encuentra en GitHub `https://github.com/jangelini/shinyAPP_geneticae`.

Capítulo 4

Resultados

4.1. Paquete de R *geneticae*

Para la construcción del paquete geneticae se seleccionaron datos abiertos de investigaciones agrícolas, se programaron y compilaron funciones, se documentaron los datos y las funciones, y se realizó el empaquetado.

4.1.1. Datos incluidos en el paquete geneticae

4.1.2. Funciones incluidas en el paquete

4.1.3. Documentación de las funciones

4.1.4. Documentación de los datos

4.1.5. Chequeo del paquete

4.1.6. *geneticae* al repositorio de R

4.1.7. Preparación del paquete y manual en windows

4.1.8. Publicación de geneticae

4.1.9. Ejemplos utilizando el paquete

4.2. Geneticae Shiny Web App

En primer lugar se debe cargar el paquete Shiny como primera línea del script:

```
library(shiny)
```

La funcion ui contiene todas las indicaciones para construir la interfaz del usuario. Estas instrucciones se pueden agrupar con respecto a los siguientes aspectos:

1. La estructura de la aplicación: Por defecto, las aplicaciones hechas con Shiny tienen un título, un panel lateral y un panel principal que se indican con las funciones headerPanel(), sidebarPanel() y mainPanel().

2. Los inputs: La reactividad de la aplicación toma como punto de partida los inputs que son los campos en los que dejamos libertad al usuario para elegir diferentes valores a través de los widgets. Hay diferentes tipos de widgets como los que reciben valores numéricos, texto, listas desplegables, etc. En nuestra aplicación, hemos incluido el widget sliderInput() que inserta una barra deslizable y permite elegir un valor de r entre -1 y 1. El valor seleccionado pasará a server.R bajo el nombre de r$input donde el identificador r.ªparece como el primer argumento de la función sliderInput().

3. Los outputs: La reactividad de la aplicación fructifica en los outputs que son los resultados (valores numéricos, tablas, gráficos) que recibe la interfaz desde el server.R. En nuestro caso, el resultado es un gráfico y se inserta con la función plotOutput().

Además de las funciones citadas, el usuario puede encontrar las siguientes:

1. h5(): Contenido de texto con diferentes tamaños. Otros tamaños son h1(), h2(), h3() y h4().

2. p(): Bloques de texto con diferentes componentes.

3. img(): Imagen (los archivos de las imágenes incluidas deben estar dentro del subdirectorio www).

El archivo server.R realiza las operaciones necesarias hasta obtener los outputs que envía como resultado a ui.R. Como hemos mencionado anteriormente, nuestra aplicación depende del valor del input r$input. Este archivo comienza de nuevo cargando el paquete Shiny y todos los necesarios para realizar los cálculos correspondientes. A excepción de las funciones definidas en R que sean necesarias

para el tratamiento de los inputs, los cálculos concretos que deben reaccionar.ª las decisiones de los usuarios están incluidos dentro de

Las funciones ui y server de nuestra aplicación se encuentra en el apéndice B.

Apéndice A

Hoja de referencia Shiny

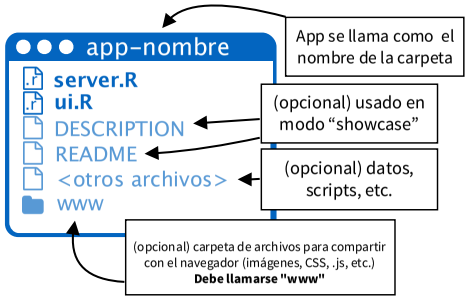
Shiny Hoja de referencia

lee mas en shiny.rstudio.com

Shiny 0.10.0 Actualizado: 6/14

RStudio

1. Estructura Cada app es una carpeta que contiene un archivo **server.R** y comúnmente un archivo **ui.R** (opcionalmente contiene archivos extra)



funciones render*

function	espera	crea
renderDataTable	objetos como tablas	tabla DataTables.js
renderImage	lista atributos imagenes	imagen HTML
renderPlot	gráfica	gráfica
renderPrint	salida impresa	texto
renderTable	objetos como tablas	tabla simple
renderText	cadena de caracteres	texto
renderUI	objeto "tag" o HTML	elemento UI (HTML)

valores de entrada (input) son reactivos.
Deben estar rodeados por uno de:

render* - crea un componente shiny UI (interfaz)
reactive - crea una expresión reactiva
observe - crea un observador reactivo
isolate - crea una copia no-reactiva de un objeto reactivo

2. server.R Instrucciones que constituyen los componentes R de tu app. Para escribir server.R:

Ⓐ Provee server.R con el mínimo de código necesario, **shinyServer(function(input, output) {})**.

Ⓑ Define los componentes en R para tu app entre las llaves { } después de **function(input, output)**.

Ⓒ Guarda cada componente R destinados para tu interfaz (UI) como **output$<nombre componente>**.

Ⓓ Crea cada componente de salida con una función *render**.

Ⓔ Dale a cada función render* el código R que el servidor necesita para construir el componente. El servidor notará valores reactivos que aparecen en el código y reconstruirá el componente cada vez que estos valores cambian.

Ⓕ Has referencia a valores en "widgets" con **input$<nombre del widget>**.

server.R

```
# carga paquetes, scripts, datos

shinyServer(function(input, output) {

  # crea variables específicos para usuario

    output$texto <- renderText({
      input$titulo
    })

    output$gráfica <- renderPlot({
      x <- mtcars[ , input$x]
      y <- mtcars[ , input$y]
      plot(x, y, pch = 16)
    })

})
```

3. Ejecución Coloca código en el lugar donde correrá la menor cantidad de veces

Corre una vez - código puesto *fuera de* ***shinyServer*** solo corre una vez cuando inicias tu app. Úsalo para instrucciones generales. Crea una sola copia en memoria.

Corre una vez por usuario - código puesto *dentro de* ***shinyServer*** corre una vez por cada usuario que visita tu app (o refresca su navegador). Úsalo para instrucciones que necesitas dar por cada usuario del app. Crea una copia por cada usuario.

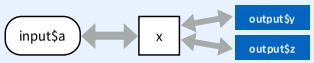
Corre a menudo - código puesto dentro de una función *render**, *reactive*, o *observe* correrá muchas veces. Úsalo solo para código que el servidor necesita para reconstruir un componente UI después de que un widget cambia.

4. Reactividad Cuando una entrada (input) cambia, el servidor reconstruye cada salida (output) que depende de ella (también si la dependencia es indirecta). Puedes controlar este comportamiento a través de la cadena de dependencias.

render* - Una salida se actualiza automáticamente cuando una entrada en su función *render** cambia.


```
output$z <- renderText({
  input$a
})
```

reactive - usa *reactive* para crear objetos que se usaran en múltiples salidas.




```
x <- reactive({
  input$a
})

output$y <- renderText({
  x()
})
output$z <- renderText({
  x()
})
```

isolate - usa *isolate* para usar una entrada sin dependencia. Shiny no reconstruirá la salida cuando una entrada aislada cambia


```
output$z <- renderText({
  paste(
   isolate(input$a),
   input$b
  )
)
```

observe - usa *observe* para crear código que corre cuando una entrada cambia, pero que no crea un objeto de salida.


```
observe({
  input$a
  # código para correr
})
```


Traducido por Frans van Dunné • innovateonline.nl

ui.R

```
A shinyUI(fluidPage(

    titlePanel("datos mtcars"),
  B sidebarLayout(
      sidebarPanel(

    C   textInput("titulo", "titulo gráfica:",
          value = "x v y"),

        selectInput("x", "Escoge una var x:"
          choices = names(mtcars),
          selected = "disp"),

        selectInput("y", "Escoge una var y:"
          choices = names(mtcars),
          selected = "mpg")
      ),

      mainPanel(
        h3(textOutput("texto")),
        plotOutput("plot")
      )
    )
))
```

C En cada panel o columna pon…

Componentes R - Los objetos de salida que has definido en **server.R**. Para colocar un componente:

1. Selecciona la función ***Output*** que construye el tipo de objeto que quieres colocar en la UI.
2. Pasa la función ***Output*** a una cadena de caracteres correspondiente al nombre del objeto en **server.R**:

output$gráfica <- renderPlot({ ... }) ⟷ plotOutput("gráfica")

funciones *Output

dataTableOutput	tableOutput
htmlOutput	textOutput
imageOutput	uiOutput
plotOutput	verbatimTextOutput

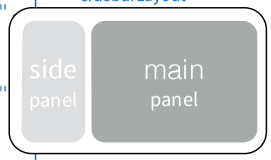
5. ui.R Una descripción de la interfaz (UI) de tu app, la página web que muestra tu app.

Para escribir ui.R:

A Incluye el mínimo de código necesario para ui.R, **shinyUI(fluidPage())**
* nota: usa **navbarPage** en vez de **fluidPage** si quieres que tu app tenga múltiples páginas conectados con un navbar

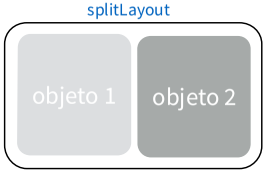
B Construye el plano para tu UI. **sidebarLayout** da una composición estándar cuando se usa con **sidebarPanel** y **mainPanel**. **splitLayout**, **flowLayout**, e **inputLayout** dividen la página en regiones equidistantes. **fluidRow** y **column** trabajan juntos para crear planos basados en rejillas, que puedes usar para componer una página o panel.

sidebarLayout



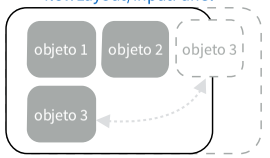

```
shinyUI(fluidPage(
  sidebarLayout(
    sidebarPanel(…),
    mainPanel(…)
  )
))
```

splitLayout



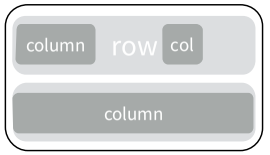

```
shinyUI(fluidPage(
  splitLayout(
    numericInput(…),
    selectInput(…)
  )
))
```

flowLayout/inputPanel




```
shinyUI(fluidPage(
  flowLayout(
    numericInput(…),
    selectInput(…),
    sliderInput(…)
  )
))
```

fluidRow




```
shinyUI(fluidPage(
  fluidRow(
    column(width = 4, …),
    column(width = 2,
      offset = 3, …),
  ),
  fluidRow(
    column(width = 12, …)
  )
))
```

Widgets - El primer argumento de cada función de *widget* es el **<nombre>** del widget. Puedes acceder a su valor actual en **server.R** con **input$<nombre>**

widget	función	argumentos comunes
Botón de acción	`actionButton`	inputId, label
casilla	`checkboxInput`	inputId, label, value
grupo de casillas	`checkboxGroupInput`	inputId, label, choices, selected
selección de fechas	`dateInput`	inputId, label, value, min, max, format
selección rango fechas	`dateRangeInput`	inputId, label, start, end, min, max, format
subir archivo	`fileInput`	inputId, label, multiple
campo numerico	`numericInput`	inputId, label, value, min, max, step
botón de selección	`radioButtons`	inputId, label, choices, selected
casilla de selección	`selectInput`	inputId, label, choices, selected, multiple
deslizador	`sliderInput`	inputId, label, min, max, value, step
botón de envío	`submitButton`	text
campo de texto	`textInput`	inputId, label, value

HTML

Elementos HTML - Añade elementos html con funciones shiny similares a etiquetas HTML comunes.

a	tags$col	tags$form	tags$input	tags$output	tags$sub
tags$abbr	tags$colgroup	h1	tags$ins	p	tags$summary
tags$address	tags$command	h2	tags$kbd	tags$param	tags$sup
tags$area	tags$data	h3	tags$keygen	pre	tags$table
tags$article	tags$datalist	h4	tags$label	tags$progress	tags$tbody
tags$aside	tags$dd	h5	tags$legend	tags$q	tags$td
tags$audio	tags$del	h6	tags$li	tags$ruby	tags$textarea
tags$b	tags$details	tags$head	tags$link	tags$rp	tags$tfoot
tags$base	tags$dfn	tags$header	tags$mark	tags$rt	tags$th
tags$bdi	div	tags$hgroup	tags$map	tags$s	tags$thead
tags$bdo	tags$dl	hr	tags$menu	tags$samp	tags$time
tags$blockquote	tags$dt	HTML	tags$meta	tags$script	tags$title
tags$body	em	tags$i	tags$meter	tags$section	tags$tr
br	tags$embed	tags$iframe	tags$nav	tags$select	tags$track
tags$button	tags$eventsource	img	tags$noscript	tags$small	tags$u
tags$canvas	tags$fieldset	includeCSS	tags$object	tags$source	tags$ul
tags$caption	tags$figcaption	includeMarkdo	tags$ol	span	tags$var
tags$cite	tags$figure	wn	tags$optgroup	strong	tags$video
code	tags$footer	includeScript	tags$option	tags$style	tags$wbr

6. Corre tu app

runApp - corre archivos locales
runGitHub - corre archivos alojados en www.GitHub.com
runGist - corre archivos guardados como gist (gist.github.com)
runURL - corre archivos guardado en algún URL

7. Comparte tu app Presenta tu app como una página web accesible en linea

ShinyApps.io

Aloja tus apps en el servidor de RStudio. Opciones gratis y pagas.
www.shinyapps.io

Shiny Server

Construye to propio servidor linux para alojar apps. Gratis y de código abierto.
shiny.rstudio.com/deploy

Shiny Server Pro

Construye un servidor comercial con autenticación, gestión de recursos y más.
shiny.rstudio.com/deploy

Apéndice B

Guías para usuario de Geneticae APP

Apéndice C

Código R de Geneticae APP