FACULTAD DE CIENCIAS AGRARIAS
UNIVERSIDAD NACIONAL DE ROSARIO

Paquete de R y aplicación Web para el análisis de datos provenientes de ensayos multiambientales

JULIA ANGELINI

TRABAJO FINAL PARA OPTAR AL TITULO DE
ESPECIALISTA EN BIOINFORMÁTICA

DIRECTOR: Gerardo Cervigni
CO-DIRECTOR: Marcos Prunello

AÑO: 2019

Paquete de R y aplicación Web para el análisis de datos provenientes de ensayos multiambientales

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

A lo largo de la historia de la agricultura, el hombre ha desarrollado el mejoramiento vegetal (o fitomejoramiento) en forma sistemática y lo ha convertido en un instrumento esencial para la mejora de la producción agrícola en términos de cantidad, calidad y diversidad.

La humanidad depende, directa o indirectamente, de las plantas. Para la alimentación, ya que todos sus alimentos son vegetales o se derivan de éstos por ejemplo: carne, huevos y productos lácteos. De las plantas se deriva también la mayoría de las fibras textiles, fármacos, combustibles, lubricantes y materiales de construcción.

El fitomejoramiento, en un sentido amplio, es el arte y la ciencia de alterar o modificar la herencia de las plantas para obtener cultivares (variedades o híbridos) mejorados genéticamente, adaptados a condiciones específicas, de mayores rendimientos económicos y de mejor calidad que las variedades nativas o criollas [1]. En otras palabras, el fitomejoramiento busca crear plantas cuyo patrimonio hereditario esté de acuerdo con las condiciones, necesidades y recursos de los productores rurales, de la industria y de los consumidores, o sea de todos aquellos que producen, transforman y consumen productos vegetales.

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.

Generalmente, en etapas tempranas de estos programas existen un gran núme-

ro de genotipos experimentales con pocos antecedentes de evaluación; mientras que en etapas posteriores se trabaja con pocos genotipos altamente selectos.

La utilización adecuada de procedimientos de análisis de datos agronómicos y ambientales es una condición inherente al desarrollo actual y futuro de investigaciones orientadas a mejorar los cultivos en forma económica y ambientalmente sustentable. En etapas avanzadas de los programas de mejoramiento, los ensayos multiambientales (EMA) que comprenden experimentos en múltiples ambientes son herramientas fundamentales para incrementar la productividad y rentabilidad de los cultivos. Estos son frecuentes en investigaciones agricolas de comparación de rendimiento, ya que constituyen una de las principales estrategias de identificación de genotipos vegetales superiores y de ambientes en los cuales estos se expresan de manera diferencial.

Debido a que las regiones de producción de los principales cultivos cubren áreas ecológicas muy extensas, se observan variaciones en las condiciones climáticas y de suelo. Por lo tanto, la aparición de la interacción genotipo ambiente (IGA) es inevitable, provocando respuestas altamente variables en los diferentes ambientes (Crossa et al., 1990; Cruz Medina, 1992; Kang y Magari, 1996). 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 mejoramiento.

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 ".

Peto (1982) ha distinguido las interacciones cuantitativas, llamada tambien sin cambio de rango (COI), o *no crossover*, de las interacciones cualitativas, denominada tambien con cambio de rango (COI) o *crossover*(Cornelius et al., 1996). Cuando dos genotipos X e Y tienen una respuesta diferencial en dos ambientes diferentes, pero su ordenación permanece sin cambios se dice que la IGA es *no crossover* (Figura 1(A)). Sin embargo, 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.

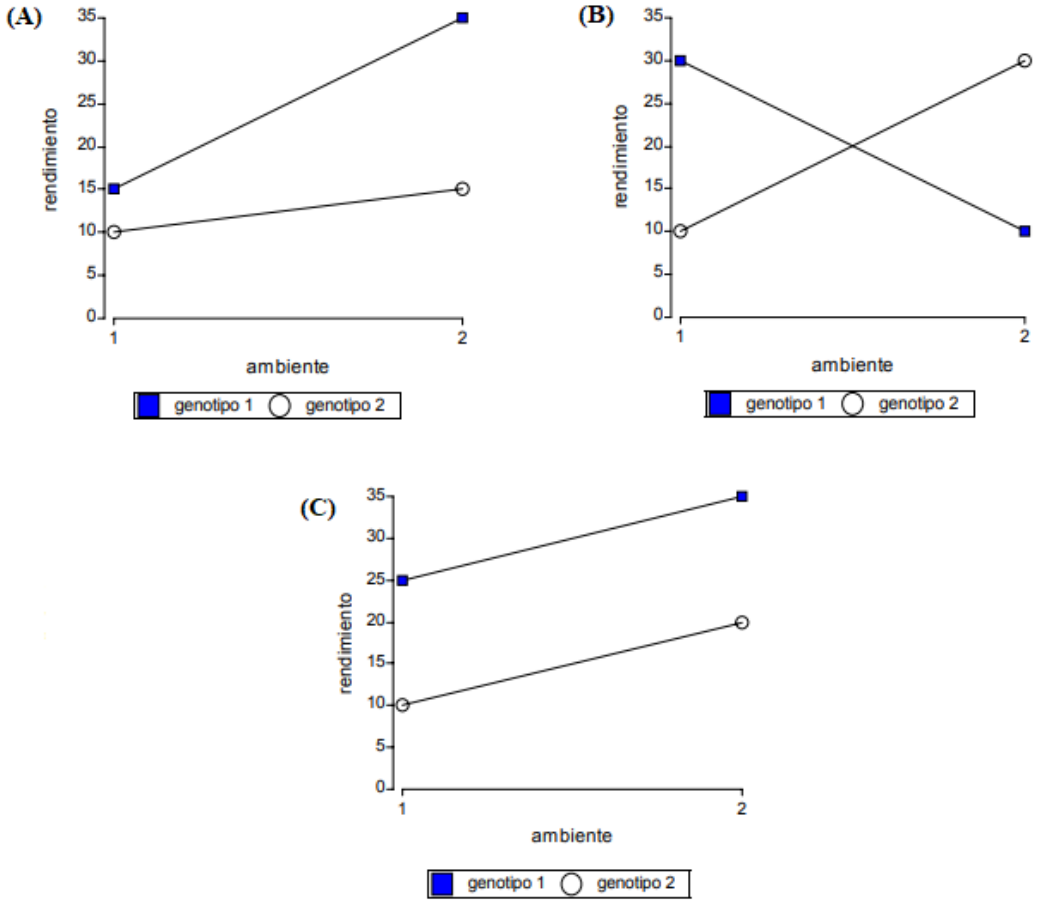


Figura 1.1: Representación gráfica de tipos de IGA: (A)IGA no crossover, (B) IGA crossover y (C) no IGA

Distintos conceptos como regiones ecológicas, ecotipos, mega-ambientes, adaptaciones de germoplasma tanto en sentido amplio (a través de los ambientes) como específico (para cada ambiente o grupos de ambiente particular) (Kang et

al., 2004) se pueden analizar a partir de la interacción genotipo-ambiente (Yan y Hunt, 2001).

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. La metodología estadística más difundida para analizar los datos provenientes de EMA se basa en modificaciones de los modelos de regresión, análisis de variancia (*Analysis of Variance*, ANOVA) y técnicas de análisis multivariado.

Particularmente, para el estudio de la interacción y los análisis que de ella se derivan, dos modelos multiplicativos han aumentado su popularidad entre los fitomejoradores como una herramienta de análisis gráfico: el modelo de los efectos principales aditivos y interacción multiplicativa (AMMI, *Additive Main effects and Multiplicative Interaction*) (Kempton 1984, Gauch, 1988), y el de regresión por sitio (SREG) (Cornelius et al., 1996; Crossa y Cornelius, 1997 y 2002). 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. A través del modelo AMMI se obtiene el gráfico biplot GE el cual es usado para explorar patrones puramente atribuibles a los efectos GE. Para el modelo SREG, Yan y Hunt (2002), presentaron la técnica GGE biplot usado para explorarnsimultáneamente patrones de variación en la suma G+GE.

Una limitación importante de la mayoría de las propuestas de análisis provenientes de EMA es que requieren que el conjunto de datos este completo. Aunque los EMA están diseñados para que todos los genotipos se evalúen en todos los ambientes, las tablas de datos genotipo x ambiente completas son poco frecuentes (no todos los genotipos se encuentran en todos los ambientes). Esto ocurre, por ejemplo, debido a errores de medición o causas naturales como, por ejemplo, la destrucción de plantas por animales, inundaciones o durante la cosecha, la incorporación de nuevos genotipos y a que otros se descartan por su pobre desempeño (Hill y Rosemberg, 1985). En estos casos, entre las posibles soluciones para tratar con una tabla de datos incompleta es (i) el uso de un subconjunto completo de

datos, eliminando aquellos genotipos que tienen valores faltantes (Ceccarelli et al., 2007, Yan et al., 2011), (ii) completar datos faltantes con la media ambiental, o (iii) imputación de datos faltantes con valores estimados utilizando, por ejemplo, un modelo multiplicativo (Kumar et al., 2012).

En este contexto, el análisis de datos provenientes de EMA 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 contemple la mayoría de las rutinas necesarias para analizar los datos provenientes de EMA. Este problema, puede ser resulto a partir del software R. Se trata de un proyecto de software libre distribuido bajo los términos de la *General Public Licence* (GNU) que surge como resultado de la implementación de uno de los lenguajes más utilizados en investigación por la comunidad estadística, el lenguaje S. A diferencia de los programas estadísticos utilizados frecuentemente, R no dispone de una interfaz gráfica lo cual genera dificultad en su uso para aquellos que no se encuentran familiarizados con el uso de un lenguaje de programación. Sin embargo, brinda mayores posibilidades en cuanto a la manipulación y análisis de los datos ya lw permite a los usuarios definir sus propias funciones y pesonalizar el tipo de análisis que desean realizar.

R forma parte de un proyecto colaborativo ya que promueve el hecho de que los usuario creen funciones y las ponga al alcance de toda la comunidad. Sin embargo, como muchas veces no resulta sencillo reutilizar una función creada por algun usuario se ha introducido la posibilidad de crear paquetes (*package*) o librerías. Estas son 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.

R cuenta con 14 paquetes básicos y 29 recomendados para su funcionamiento instalados automaticamente en él, como por ejemplo, *base* o *stats*. Dado que la comunidad de usuarios que programan en R ha ido creciendo notablemente en los últimos años y que muchos de ellos han ido proporcionando librerías, se

cuenta con una gran cantidad de paquetes que extienden las funciones básicas de R. Entre ellos 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 paquetes oficiales, existen otros 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 programas que tienen 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* que permite desarrollar aplicaciones Web utilizando R, acercando 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 los datos provenientes de EMA, incluyendo además metodología recientemente publicada que no se encuentra disponible en R; (ii) crear una interfaz gráfica, entre R y el usuario, mediante Shiny con el fin de poder realizar los análisis disponibles en el paquete creado sin necesidad de utilizar el lenguaje de programación.

Pensar en hacer algun comentario de R studio

[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

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. Métodos estadísticos

3.1.1. Modelo AMMI

Modelo AMMI clásico

El modelo AMMI, *Additive Main effects and Multiplicative Interaccion* llamados así por Zobel et al (1998) y Gauch (1998) es un modelo multiplicativo en el cual se expresa la respuesta de un genotipo en un ambiente de la siguiente forma:

$y_{ij} = \mu + G_i + A_j + \sum_{k=1}^{q} \lambda_k \alpha_{ik} \gamma_{jk}$ $i = 1, ..., g$; $j = 1, ..., a$ $q = min(g-1, a-1)$

donde

- y_{ij} es la característica fenotípica evaluada (rendimiento u otra variable cuantitativa de interés) del i-ésimo genotipo en el j-ésimo ambiente,
- μ es la media general,
- G_i es el efecto del i-ésimo genotipo,
- A_j es el efecto del j-ésimo ambiente
- $\sum_{k=1}^{q} \lambda_k \alpha_{ik} \gamma_{jk}$ es la sumatoria de componentes multiplicativas utilizadas para modelar la IGA. Siendo, λ_k el valor singular para la k-ésima IPC α_{ik} y γ_{jk} son los scores de las PC para el i-ésimo genotipo y el j-ésima ambiente para la k-ésima componente, respectivamente;

La estimación de los parámetros de interacción GA en el modelo AMMI de efectos fijos se hace por medio de la DVS de una matriz que contiene los residuos del modelo aditivo luego de ajustar por mínimos cuadrados el modelo de efectos principales.

Generalmente los dos primeros términos multiplicativos son suficientes para explicar los patrones de interacción; la variabilidad remanente se interpreta como ruido.

Para visualizar gráficamente los patrones de interacción, se construyen los biplots *Genotipe-Environment*, a menudo llamados biplots GE. El concepto del biplot fue presentado por Gabriel (1971). Es una representación, en un mismo gráfico, de las filas (individuos) y las columnas (variables) de una matriz de datos.

Biplot GE

En los modelos AMMI, el biplot GE ayuda a interpretar la variación producida por los efectos de la IGA. Se grafican en un sistema de coordenadas cartesianas de dos dimensiones los scores de los genotipos (α_{ik}) y los ambientes (γ_{jk}), ponderados por la raíz cuadrada del autovalor correspondiente (λ_k).

Tanto los genotipos y los ambientes son definidos como vectores desde el origen (0,0) a los extremos por sus scores, por lo que el biplot se interpreta en términos de las direcciones de vectores y sus proyecciones.

La magnitud del vector de un ambiente da una idea de la contribución del mismo a la interacción. Los puntos de los genotipos que están cercanos al origen se interpretan como que contribuyen poco a la interacción, es decir, se adaptan de igual manera a todos los ambientes. Puntos cercanos entre sí tienen patrones de interacción similares, mientras que puntos alejados entre sí tienen patrones diferentes. Cuando los marcadores (o puntos) de los ambientes y genotipos están próximosn(entre ellos forman un ángulo $< 90^o$ ó $> 270^o$), indica que contribuyen positivamente a la interacción, esndecir, hay una asociación positiva entre ese genotipo y ese ambiente; y mientras más alejados delnorigen estén los marcadores, más fuerte será esa asociación. Una fuerte asociación positiva entre un genotipo y un ambiente se interpreta como que ese ambiente es muy favorable para ese genotipo. De manera similar, cuando los marcadores del genotipo y el ambiente están opuestos entre sí (forman un ángulo entre 90^o y 270^o) se interpreta que ese ambiente es muy desfavorable para ese genotipo.
En la Figura 1, se presenta un ejemplo de un biplot GE con 6 ambientes (A, B, C,

D, E y F) y 10 genotipos (1, 2, 3, 4, 5, 6, 7, 8, 9 y 10). Se observa que la magnitud de los vectores de los ambientes A y E es mayor a la de los demás ambientes, es decir que esos dos ambientes son los que más contribuyen a la interacción. La cercanía de los marcadores de los genotipos 1 y 2 indica que esos genotipos tienen patrones de interacción similares, y a la vez, muy distintos a los del genotipo 4. Del biplot también se destacan las cercanías entre el genotipo 9 y el ambiente A, entre el genotipo 5 con el ambiente C, entre los genotipos 1 y 2 con el ambiente E, entre los genotipos 6, 7 y 10 con el ambiente F y entre el genotipo 4 con el ambiente D, lo que indica, debido a la gran distancia al origen, una fuerte asociación positiva entre esos genotipos y esos ambientes, es decir, esos ambientes son muy favorables para esos genotipos.

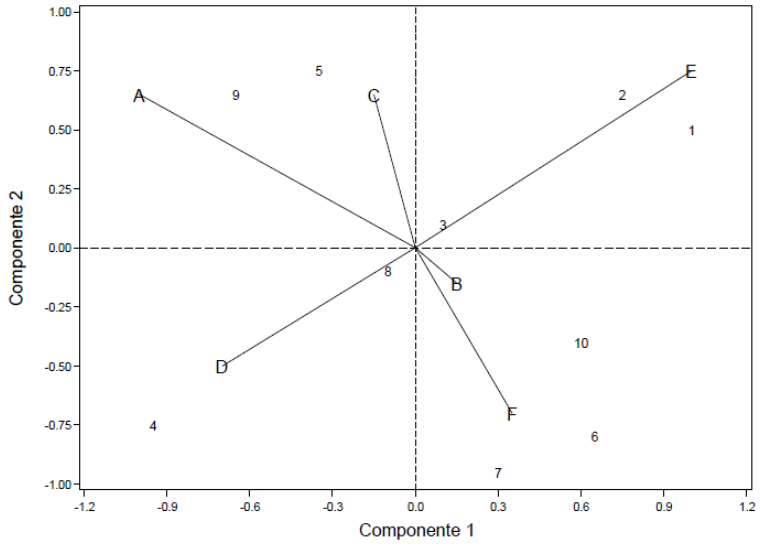


Figura 3.1: Ejemplo de un biplot GE

Entre las altas asociaciones negativas se puede mencionar a la del ambiente A con el genotipo 10 (marcadores opuestos en el biplot) y se interpreta que ese ambiente es considerablemente desfavorable para ese genotipo.
También se observa que los genotipos 3 y 8 están próximos al origen, lo que quiere decir que se adaptan en igual medida a todos los ambientes.

Modelo AMMI robusto

3.1.2. Modelo SREG

El modelo SREG (Cornelius et al., 1996; Crossa y Cornelius, 1997 y 2002) expresa el rendimiento medio de un genotipo en un ambiente en función del efecto ambiente aditivo y los efectos genotipo e interacción agrupados y en forma multiplicativa.

$y_{ij} = \mu + A_j + \sum_{k=1}^{q} \lambda_k \alpha_{ik} \gamma_{jk} \; i = 1, ..., g; \; j = 1, ..., a \; q = min(g-1, a-1)$

donde

- y_{ij} es la característica fenotípica evaluada (rendimiento u otra variable cuantitativa de interés) del i-ésimo genotipo en el j-ésimo ambiente,
- μ es la media general,
- G_i es el efecto del i-ésimo genotipo,
- A_j es el efecto del j-ésimo ambiente
- $\sum_{k=1}^{q} \lambda_k \alpha_{ik} \gamma_{jk}$ es la sumatoria de componentes multiplicativas utilizadas para modelar los efectos G e IGA en forma conjunta. Siendo, λ_k el valor singular para la k-ésima IPC α_{ik} y γ_{jk} son los scores de las PC para el i-ésimo genotipo y el j-ésima ambiente para la k-ésima componente, respectivamente;

Para visualizar conjuntamente estos dos efectos, con remoción de los efectos de ambiente (datos centrados por sitio), Yan et al. (2000) proponen los gráficos biplots GGE (Genotipe plus Genotipe-Environment). A partir de estos gráficos se puede investigar la diferenciación de mega-ambientes entre los ambientes en estudio, seleccionar cultivares superiores en un mega-ambiente dado y seleccionar los mejores ambientes de evaluación para analizar las causas de la interacción GA. Se define como mega-ambiente a un grupo de ambientes en donde los cultivares de mejor desempeño son los mismos.

Biplot GGE En los modelos SREG, el biplot GGE, ayuda a interpretar conjuntamente la variación producida por los efectos principales de los G e IGA.

Para la construcción de los biplots GGE, al igual que para los biplots GE, se grafican en un sistema de coordenadas cartesianas de dos dimensiones los scores de los genotipos (α_{ik}) y los ambientes (γ_{jk}), ponderados por la raíz cuadrada del autovalor correspondiente (λ_k).

Para una mejor comprensión de las interpretaciones que se pueden extraer del gráfico biplot GGE, se presenta un ejemplo del mismo para un ensayo de 6 ambientes (A, B, C, D, E y F) con 12 genotipos (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 y 12).

Para identificar los mejores genotipos en un ambiente a través del biplot GGE, Yan y Hunt (2002) sugieren trazar una recta que pase por el identificador del ambiente y el origen, la cual constituye el eje del ambiente. Luego, las proyecciones de los marcadores de los genotipos sobre ese eje, proveen un ranking de los genotipos en ese ambiente. El genotipo de mayor rendimiento en el ambiente es aquel cuya proyección sobre el eje está más alejada del origen hacia el semi-eje donde se encuentra el marcador del ambiente. Aquel cuya proyección sea la segunda más alejada del origen en ese sentido, es el de segundo mejor rendimiento y así hasta llegar al de menor rendimiento en el ambiente, que es aquel cuya proyección está más alejada del origen en sentido contrario al identificador del ambiente. La perpendicular al eje del ambiente que pasa por el origen, divide a los genotipos de rendimiento superior e inferior al promedio del ambiente.

Como se observa en la Figura 4.2, el genotipo de mayor rendimiento en el ambiente D es el 4, luego le sigue el 7, luego el 8, y así sucesivamente hasta llegar al genotipo 12, que es el de peor rendimiento en ese ambiente.

Los marcadores de los genotipos 4, 7, 8, 11, 2 y 5 quedan del lado del marcador del ambiente D, de acuerdo a la división de la perpendicular que pasa por el origen, por que se interpreta que estos genotipos tienen un rendimiento mayor al promedio del ambiente D. Los restantes genotipos tienen un rendimiento inferior al promedio.

Para visualizar el desempeño de un genotipo en los diferentes ambientes, Yan y Hunt (2002) sugieren graficar una línea que una el marcador del genotipo con el origen y luego trazar otra línea perpendicular a la primera. Esta última perpendicular es la que separa los sitos favorables y desfavorables para el genotipo. Los sitios cuyos marcadores queden en el mismo lado donde está el genotipo son los mejores para ese genotipo y los restantes son aquellos donde el genotipo rinde por debajo de su promedio.

Como se puede apreciar en la Figura 4.3, la perpendicular al marcador del genotipo 2, determina que los ambientes favorables para ese genotipo son A, E, F y D, en donde el mismo tiene rendimientos mayores a su promedio. Los ambientes desfavorables son B y C. También se observa que el ambiente más favorable es

el A, luego le siguen el E y el F. Si bien el ambiente D también es favorable, en ese ambiente el rendimiento del genotipo 2 es apenas superior a su rendimiento medio ya que el marcador del ambiente D, está muy cerca de la perpendicular que pasa por el origen.

Para comparar dos genotipos, se propone unir mediante una línea recta los genotipos a comparar, luego trazar una línea que pase por el origen y que sea perpendicular a la línea que une a los genotipos. Esta última línea es la que separa sitios favorables a uno y a otro genotipo.

Los sitios cuyos marcadores queden en el mismo lado donde está el marcador del genotipo son los mejores para ese genotipo. Si un ambiente queda posicionado sobre la línea perpendicular, los dos genotipos tienen rendimientos similares en ese ambiente. Si dos genotipos están cercanos, sus rendimientos son similares en los ambientes evaluados. Por último, si todos los ambientes quedan a un lado de la línea perpendicular, el genotipo cuyo identificador está de ese lado rinde más que el otro en todos los ambientes. En la Figura 4.4 se puede ver la comparación de los desempeños de los genotipos 6 y 8. Los ambientes que resultan favorables para el genotipo 6 son el E, el A y el F. Mientras que los favorables para el genotipo 8 son el B, el C y el D.

Para poder identificar mega-ambientes y los mejores genotipos en cada uno de ellos se propone graficar un polígono envolvente. Este polígono se forma uniendo los genotipos más extremos en el biplot con segmentos continuados. Luego se trazan líneas rectas que pasan por el origen y que son perpendiculares a cada uno de los lados del polígono (o a sus proyecciones). De esta forma, el biplot queda dividido en cuadrantes, y los sitios que quedan dentro un mismo cuadrante se consideran pertenecientes a un mismo mega-ambiente. Generalmente cada cuadrante contiene un genotipo en el vértice, que es el de mayor rendimiento en el mega-ambiente. En la Figura 4.5 se presenta el biplot GGE con el polígono envolvente y las perpendiculares a sus lados, que ayudan a la interpretación del mismo. En primer lugar se observa que los marcadores de los ambientes A y B son mayores a los restantes, lo que indica que la variabilidad en esos ambientes es superior, es decir, en ellos es donde mejor se diferencian los efectos de los genotipos. Las perpendiculares a los lados del polígono envolvente determinan tres mega-ambientes:

- uno formado por los ambientes A, E y F, en donde el genotipo de mejor desempeño es el 2 (se encuentra en el vértice del polígono encerrado por las

perpendiculares).

- otro está formado solo por el ambiente D y el genotipo ganador en él es el 4, y
- el tercer mega-ambiente lo componen los ambientes B y C, en este caso el genotipo ganador es el 1.

Si se identifican distintos mega-ambientes, la selección de genotipos debe hacerse para cada mega-ambiente en particular. Los genotipos se seleccionan en base a su desempeño y a su estabilidad a través de los ambientes. En el biplot, se puede definir un eje medio para todos los ambientes pertenecientes a un mismo mega-ambiente. Para ello se calcula la media de scores promediando los scores de la componente 1 y la componente 2 de los ambientes pertenecientes al mega-ambiente. Una vez obtenidos estos promedios, se traza una línea recta entre ese punto medio y el origen, que se denomina eje de la media de scores de ambientes. A continuación se traza una línea que pase por el origen y que sea perpendicular a la línea media de scores de ambientes. Estas dos líneas constituyen "el eje de coordenadas de ambiente medio". Las proyecciones de los marcadores de los genotipos sobre el eje de la media de scores de ambientes da un ranking de los rendimientos de los genotipos en ese mega-ambiente. A su vez la magnitud de la proyección de los marcadores de los genotipos a la perpendicular al eje de ambientes da una idea de la estabilidad. Cuanto mayor sea esta magnitud, más inestable será el genotipo. En este ejemplo se calcula el ambiente medio para un mega-ambiente, en la Figura 4.6 se presenta el gráfico del eje medio para el mega-ambiente formado por los ambientes A, E y F. El promedio de los scores de la primer componentes es 1,00 y el de la segunda es 1,133, por lo que el punto medio que determina la dirección del eje es (1; 1,133), que está graficado con un círculo en la Figura 4.6.

Como se puede observar en el biplot el orden de los genotipos (de mayor a menor rendimiento) es: 2, 4, 12, 11, 5, 6 todos ellos con rendimientos superiores al promedio, seguidos por los de rendimiento menor al promedio: el 10, 9, 7, 3 y por último el 1, el de peor rendimiento medio en ese mega-ambiente. Debido a que las proyecciones sobre el eje perpendicular al eje medio de ambiente dan una idea de la estabilidad, se observa que el genotipo 12, el 9, el 7 y el 4 son los más inestables. También se observa que el genotipo 2, además de tener el mejor rendimiento medio es de los más estables en el megaambiente.

3.1.3. Métodos de imputación

3.2. 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.

3.2.1. 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.

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.

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.

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

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. Shiny APP

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.

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 3.2.

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

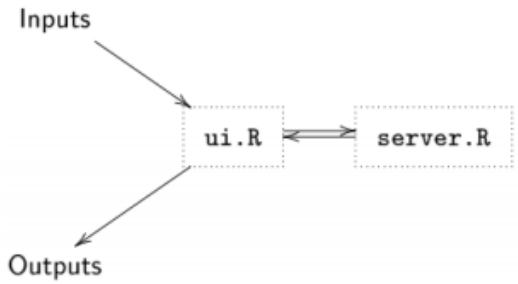


Figura 3.2: Esquema interno de la aplicación.

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.

3.3.1. 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 3.3).

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

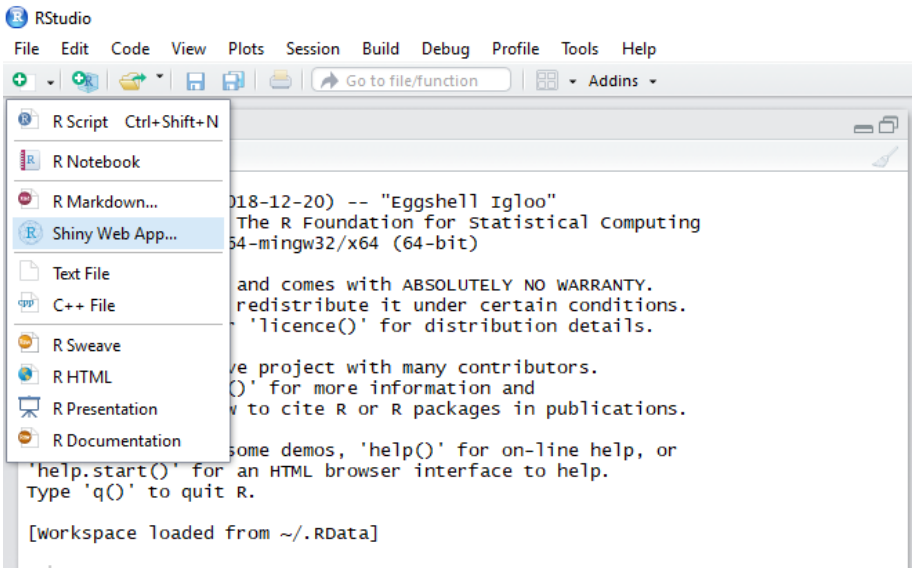


Figura 3.3: Creación de Shiny Web App desde Rstudio

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 3.4 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.

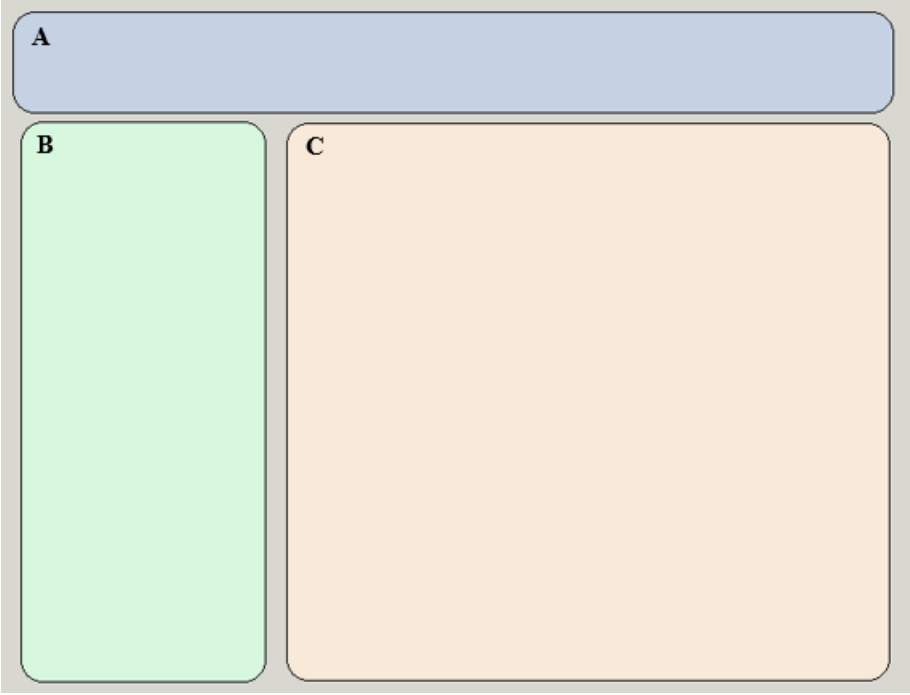


Figura 3.4: Diseño de la aplicación Shiny. **A**: título y pestañas; **B**: área de entrada; **C**: área de resultados.

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).

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		

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

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

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()	

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

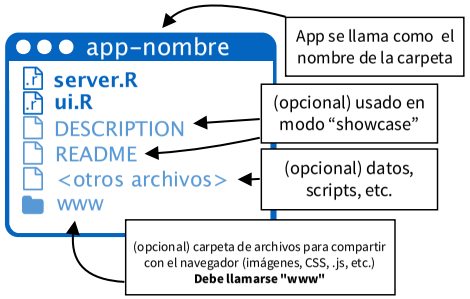
Shiny Hoja de referencia

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.

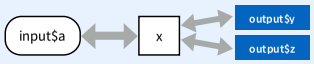
Corre a menudo - código puesto dentro de una función *render**, *reactive*, o *observe* correrá muchas veces. Úsalo solo para código que el servidor necesita para reconstruir un componente UI después de que un widget cambia.

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

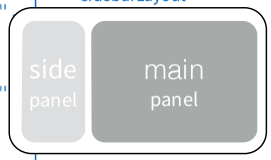
5. ui.R Una descripción de la interfaz (UI) de tu app, la página web que muestra tu app.

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

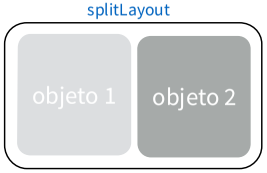
B Construye el plano para tu UI. **sidebarLayout** da una composición estándar cuando se usa con **sidebarPanel** y **mainPanel**. **splitLayout**, **flowLayout**, e **inputLayout** dividen la página en regiones equidistantes. **fluidRow** y **column** trabajan juntos para crear planos basados en rejillas,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

sidebarLayout



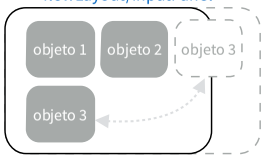

```
shinyUI(fluidPage(
  sidebarLayout(
    sidebarPanel(…),
    mainPanel(…)
  )
))
```

splitLayout



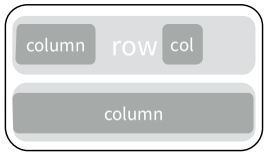

```
shinyUI(fluidPage(
  splitLayout(
    numericInput(…),
    selectInput(…)
  )
))
```

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

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

Bibliografía

[1] R.W. Allard. *Principios de la mejora genética de las plantas.* Ediciones Omega, 1967.