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 **Bioinformática**, 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 **el Centro de Estudios Fotosintéticos y Bioquímicos (CEFOBI)**, durante el período comprendido entre **los años 2017 y 2019**, bajo la dirección del **Dr. Gerardo Cervigni y Mgs. Marcos Prunello**.

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

EMA: ensayos multiambientales **IGA**: interacción genotipo ambiente **NCOI**: interacción sin cambio de rango, del inglés *no crossover interaction* **COI**: interacción con cambio de rango, del inglés *crossover interaction* **ANOVA**: análisis de la variancia, del inglés *analysis of variance* **AMMI**: modelo de los efectos principales aditivos y interacción multiplicativa, del inglés *Additive Main effects and Multiplicative Interaction* **ACP**: análisis de componentes principales **SREG**: modelo de regresión por sitio, del inlés *Site Regression model* **DVS**: descomposición de valores singulares **GNU**: *General Public Licence* **CRAN**: *Comprehensive R Archve Network* **EM**: maximización de la esperanza, del inglés *Expectation-Maximization*

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 (Allard, 1967). 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 durante 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 número 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 orienta-

das 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 (NCOI), 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 1.1(A)). Sin embargo, es de tipo *crossover* cuando hay cambios en el orden de los genotipos (Figura 1.1(B)). Cuando los genotipos responden de manera similar en ambos ambientes (Figura 1.1(C)) no hay IGA.

Distintos conceptos como regiones ecológicas, ecotipos, mega-ambientes, adaptaciones

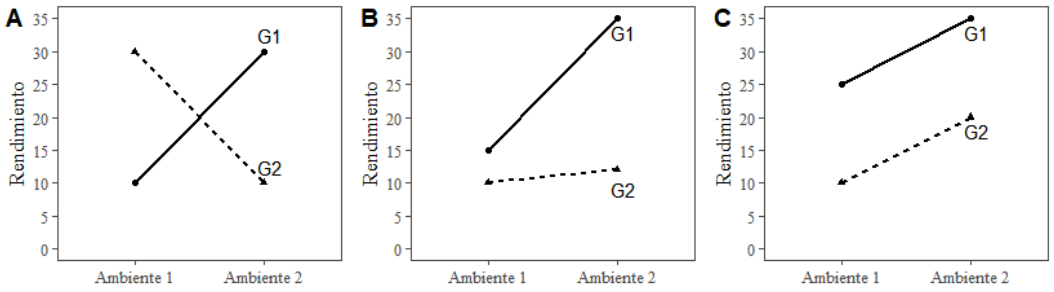


Figura 1.1: Representación gráfica de tipos de IGA: (A)IGA no crossover, (B) IGA crossover y (C) no IGA

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 (*Additive Main effects and Multiplicative Interaction*, AMMI) (Kempton 1984, Gauch, 1988), y el de regresión por sitio (*Site Regression model*, 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 (DVS) o el análisis de componentes principales (ACP) 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 *Genotipe-Environment* (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], se contaba con más de 14.000 paquetes disponibles en CRAN hasta junio de 2019. Esta gran variedad de paquetes es una de las razones por las que R tiene tanto éxito: lo más probable es que alguien ya haya resuelto un problema en el que estás trabajando, y puedes beneficiarte de su trabajo descargando su paquete.

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 un flujo de trabajo reproducible para la construcción de paquetes 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 Web 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 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 el caracter fenotípico evaluado (rendimiento o cualquier otro caracter de interes) 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 PC 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;

Los parámetros de IGA en el modelo AMMI se estiman por medio de la DVS de la 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.

Los patrones de interacción se pueden visualizar mediante los biplots GE, a menudo llamados biplots GE. El concepto del biplot fue presentado por Gabriel (1971), que consiste en una representación de las filas (individuos) y las columnas (variables) de una matriz de datos en un mismo gráfico.

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).

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

El módulo del vector de un ambiente da una idea de la contribución del mismo a la interacción. Los puntos de los genotipos que se encuentran próximos al origen indican que los mismos 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 próximos, es decir forman un aungulo $< 90^o$ ó $> 270^o$, indica que contribuyen positivamente a la interacción(hay una asociación positiva entre ese genotipo y ese ambiente); y mientras más alejados del origen se encuentre 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 3.1, 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.

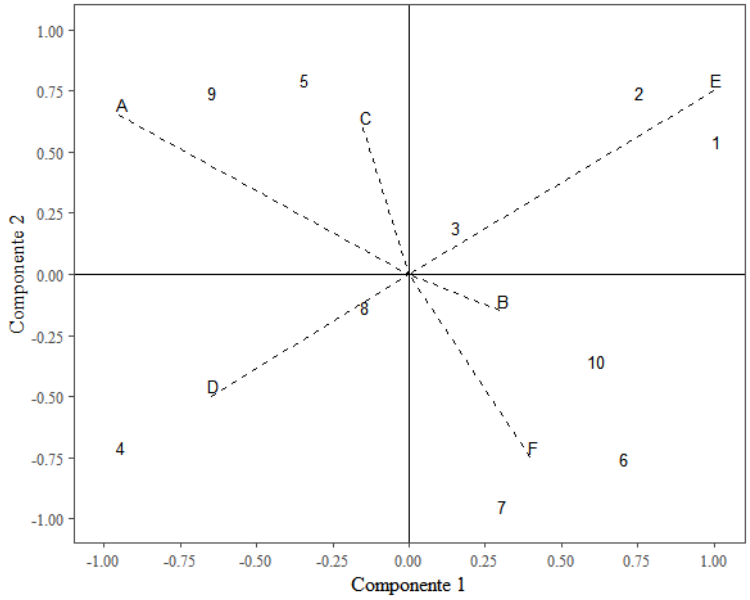


Figura 3.1: Ejemplo de un biplot GE

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

El modelo AMMI, en su forma estándar, asume que no hay valores atípicos en el conjunto de datos. La presencia de observaciones atípicas es más una regla que una excepción cuando se consideran datos agronómicos debido a errores de medición, algunas plagas / enfermedad que puede influir en algunos genotipos en un ambiente dado que resultando por ejemplo en un rendimiento inferior al esperado, o incluso debido a alguna característica inherente de los genotipos que se miden.

Rodrigues 2015 proponen una generalización robusta del modelo AMMI. La metología propuesta se puede obtener en dos etapas de la siguiente manera: primero ajustar la regresión robusta basada en el estimador M-Huber (Hub) para reemplazar el ANOVA; y luego utilizar un procedimiento DVS / APC robusto para reemplazar el DVS estándar.

En la segunda etapa, consideraron varios métodos dando lugar a total de cinco robustos: R-AMMI, H-AMMI, G-AMMI, L-AMMI, PP-AMMI.

El empleo de la versión robusta del modelo AMMI puede ser extremadamente útil debido a que una mala representación de genotipos y ambientes en los biplots puede dar como resultado un mala decisión con respecto a qué genotipos seleccionar para un conjunto dado de ambientes (es decir, megaambientes; Gauch y Zobel, 1997; Yan et al., 2000). A su vez, la elección de los genotipos incorrectos pueden provocar grandes pérdidas en términos de producción de rendimiento.

Debe tenerse en cuenta que los biplots robustos mantienen las características e interpretación estándar del modelo AMMI clásico.

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 3.2, 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 3.3, 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.

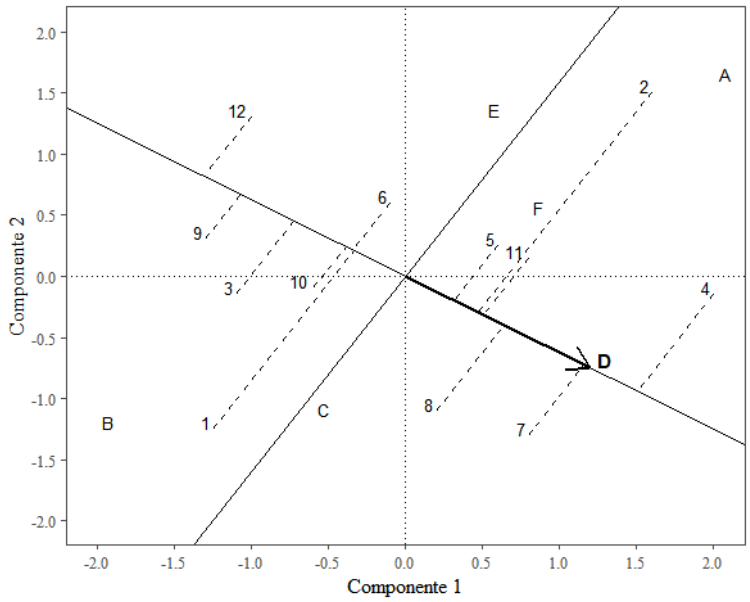


Figura 3.2: Ranking de genotipos en el ambiente D a través del biplot GGE

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 3.4 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

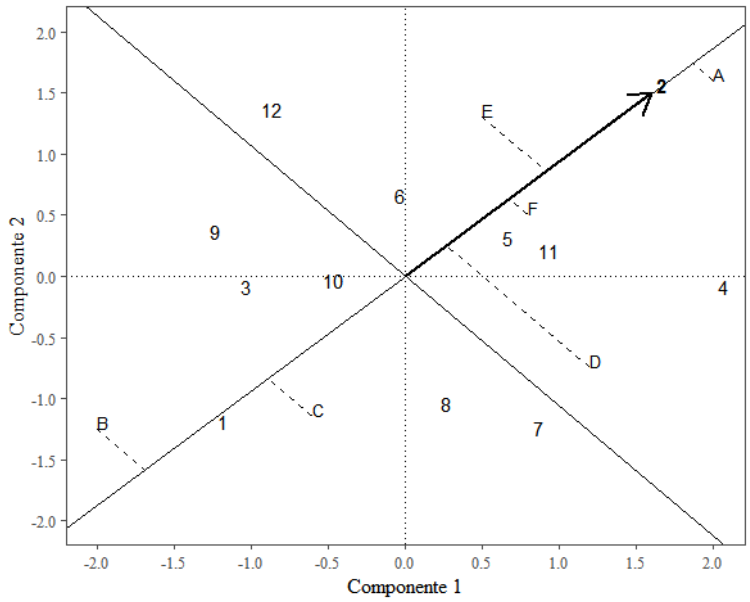


Figura 3.3: Ambientes favorables y desfavorables para el genotipo 2 en el biplot GGE

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 3.5 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

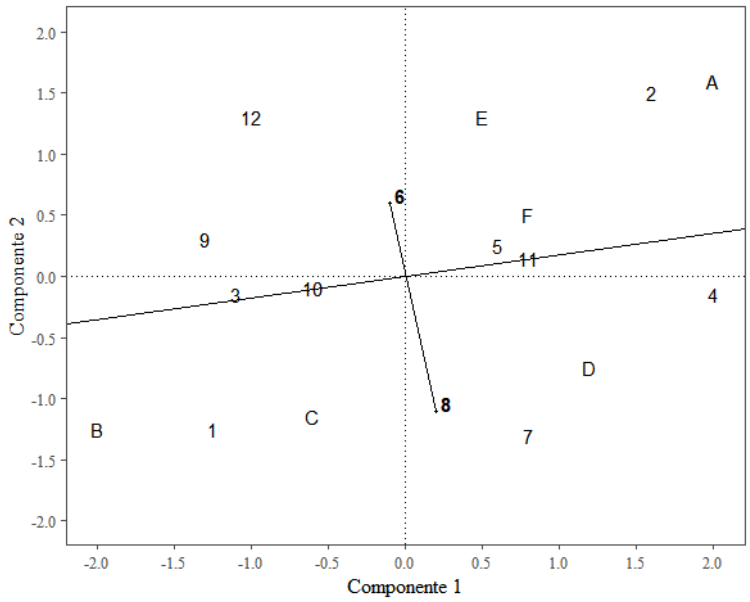


Figura 3.4: Comparación de los genotipos 6 y 8 en el biplot GGE

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 3.6.

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.

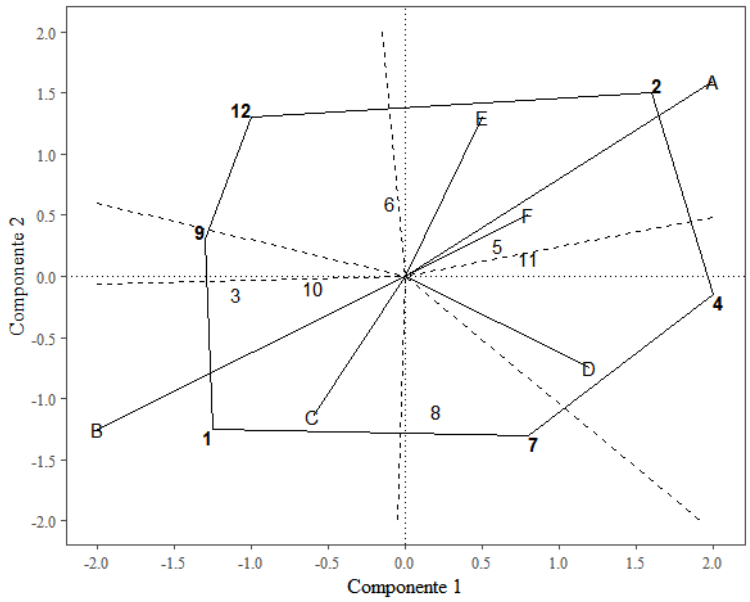


Figura 3.5: Biplot GGE con el polígono envolvente y las perpendiculares a sus lados

3.1.3. Métodos de imputación

Una limitación importante que presentan los modelos multiplicativos (AMMI y SREG) es que requieren que el conjunto de datos este completo, es decir no admiten valores perdidos. Aunque los EMA están diseñados para que todos los genotipos se evalúen en todos los ambientes, la presencia de valores faltantes es muy común, debido por ejemplo a la incorporación de nuevos genotipos, errores de medición o causas naturales como la destrucción de plantas por animales, inundaciones o durante la cosecha.

Entre las posibles soluciones para tratar un conjunto de datos con observaciones perdidas: el uso de un subconjunto completo de datos, eliminando aquellos genotipos que tienen valores faltantes (Ceccarelli et al., 2007, Yan et al., 2011), completar datos faltantes con la media ambiental, o imputación de los valores faltantes mediante estimaciones utilizando, por ejemplo, un modelo multiplicativo (Kumar et al., 2012).

Se han propuesto numerosas metodologías para superar el problema de valores ausentes en el conjunto de datos, entre las cuales se encuentran:

- EM-AMMI: Gauch y Zobel (1990) desarrollaron este enfoque mediante el cual se imputa utilizando el algoritmo de maximización de la esperanza (EM, del inglés *Expectation-Maximization*) incorporando el modelo AMMI. Consiste en un procedimiento iterativo que funciona de la siguiente forma: ..... Dependiendo del número de términos multiplicativos empleados, el método de imputación puede denominarse EM-AMMI0, EM-AMMI1, etc. (Gauch y Zobel 1990). Los estudios de

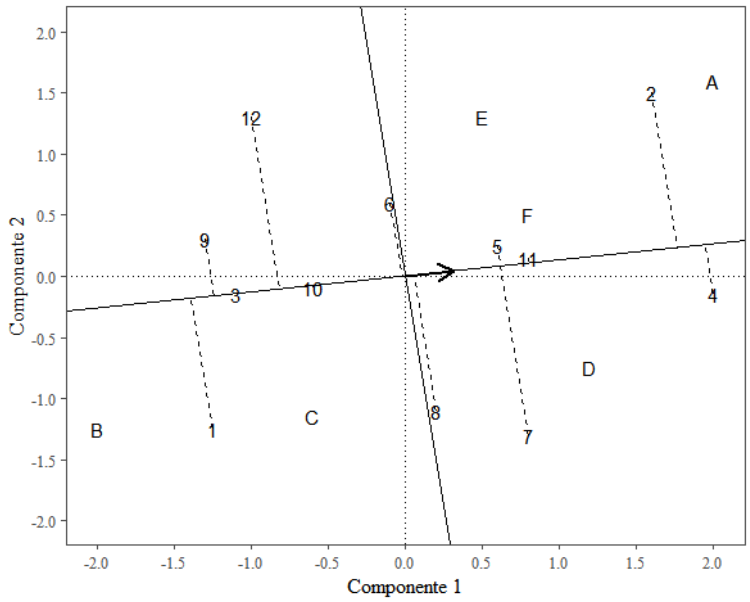


Figura 3.6: Eje de coordenadas de ambiente medio para un mega-ambiente en el biplot GGE

Caliński y col. (1992), Piepho (1995), Arciniegas-Alarcón y Dias (2009) y Paderewski y Rodrigues (2014) mostraron que se obtienen los mejores resultados para la imputación con modelos AMMI al incluir como máximo una componente multiplicativa.

- EM-SVD: Perry (2009a) propone un método de imputación que combina el algoritmo EM con DVS. Este método reemplaza los valores faltantes de una matriz $G \times E$ inicialmente por valores arbitrarios para obtener una matriz completa, y luego la DVS se calcula iterativamente en esa matriz. Al final del proceso, cuando las iteraciones alcanzan estabilidad, se obtiene la matriz imputada. .....Podria ponerlo tambien de manera formal... pero creo que con esto basta

- EM-PCA:

- Gabriel Eigen: Arciniegas-Alarcón et al. (2010) propuso un método de imputación que combina regresión y aproximación de rango inferior usando DVS. El método reemplaza inicialmente las celdas faltantes por valores arbitrarios, y posteriormente el las imputaciones se refinan a través de un esquema iterativo que define una partición de la matriz para cada valor que falta a su vez y utiliza una regresión lineal de columnas (o filas) para obtener el nueva imputación. En esta regresión, la matriz de diseño se aproxima por una matriz de menor rango usando la DVS.

- WGabriel Eigen:

Vale la pena señalar que el modelo de análisis no siempre será el mismo que el modelo 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 la ausencia de errores (de sintaxis) en la programación. Éstos son las unidades fundamentales de un código reproducible de R ya que incluyen funciones reutilizables, la documentación que describe cómo usar cada una de ellas y, además datos de ejemplo.

Los pasos necesarios para la creación de un paquete son:

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

Para la cración del paquete se utilizan numerosas funciones incluidas en el paquete *devtools* en el cual se encuentran un conjunto de paquetes que admiten varios aspectos del desarrollo de paquetes. Por lo tanto, antes de comenzar a crear el paquete se deben instalar el mismo como se indica a continuación:

```
# Instalar el paquete devtools desde CRAN
install.packages("devtools")

# Instalar el paquete devtools desde GitHub:
devtools::install_github("r-lib/devtools")
```

3.2.1. Esqueleto y estructura del paquete

Para crear la estructura del paquete se utiliza la función create_package(). El principal y único argumento requerido por dicha función es el directorio donde el nuevo paquete se alojará. Por lo general, si el directorio se llama "geneticae", entonces el nombre del paquete también será "geneticae":

```
# Cargar la libreria devtools
library(devtools)

# Crear el paquete geneticae
create_package("C:/Users/Julia/Desktop/geneticae")
```

El resultado de ejecutar la función create_package() es un paquete con los siguientes componentes:

- Un directorio R/.
- DESCRIPTION, un archivo simple cuyo objetivo es almacenar metadatos importantes sobre el paquete, epecifica el título, la versión del mismo, identifica al autor y brinda un mail de contacto, una breve descripción del paquete, la lista de los paquetes que el paquete creado necesita para funcionar, la licencia, entre otros.

  El contenido básico en un archivo DESCRIPTION es:

```
Package: geneticae
Title: What the Package Does (One Line, Title Case)
Version: 0.0.0.9000
Authors@R:
    person(given = "First",
           family = "Last",
           role = c("aut", "cre"),
           email = "first.last@example.com",
           comment = c(ORCID = "YOUR-ORCID-ID"))
Description: What the package does (one paragraph).
License: What license it uses
Encoding: UTF-8
LazyData: true
```

- Un archivo NAMESPACE

También puede incluir un archivo de proyecto de RStudio pkgname.Rproj, que hace que su paquete sea fácil de usar con RStudio; .Rbuildignore enumera los archivos que se

necesitan, pero que no deben incluirse al compilar el paquete R desde la fuente; .gitignore anticipa el uso de Git e ignora algunos archivos estándar detrás de escena creados por R y RStudio.

3.2.2. Creación de funciones y conjuntos de datos

Una vez creada la estructura del paquete se debe comenzar a incluir las funciones que contendrá. Cada una de ellas debe ser guardada en un archivo de extensión .R, en el subdirectorio R/. La funcion use_r() crea y/o abre un script de la carpeta R/.

Una vez creada una función, para probarla se utiliza load_all() que simula el proceso de construcción, instalación y conexión del paquete. Permite que las funciones creadas estén disponible rápidamente para uso interactivo, del mismo modo que si se hubiera construido e instalado el paquete y luego cargado a través de library(geneticae).

Muy frecuentemente se utilizan funciones de otros paquetes, para ello se utiliza la función use_package() que agrega el paquete a lista de los paquetes que el paquete creado necesita para funcionar del archivo DESCRIPTION.

A menudo es útil incluir datos en un paquete, con estos se proporcionan casos de uso convincentes para las funciones del paquete. La ubicación más común para los datos es data/, siendo cada archivo de este directorio un .RData que sólo contiene un objeto. Para esto, se utiliza la funció usethis::use_data(). Se puede observar que el archivo DESCRIPTION creado con la función create_package() contiene el campo LazyData: true, lo cual genera que los conjuntos de datos no ocuparán ninguna memoria hasta que los use.

3.2.3. Documentación

La documentación es uno de los aspectos más importantes del código, sin ella, los usuarios no sabrán cómo usar el paquete. Existen múltiples formas de documentar un paquete, la forma estándar es 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, convierte los comentarios con formato especial en archivos .Rd. Esta última proporciona una serie de ventajas sobre la estándar:

- 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 el archivo NAMESPACE. El flujo de trabajo para crear la documentación con el paquete *roxygen2* es el siguiente:

- Agregar comentarios a los archivos .R, los cuales comienzan con #'y preceden a una función. La primera oración se convierte en el título y el segundo párrafo es una descripción de la función. Seguidamente, las funciones son documentadas, la mayoría de las funciones tienen tres etiquetas: @param, @examples y @return.

    - @param describe los parámetros de la función, indica de que clase es el parámetro y para que sirve.
    - @examples proporciona un código ejecutable que muestra cómo usar la función en la práctica.
    - @return describe el resultado de la función.

- Ejecutar devtools::document() para convertir los comentarios de roxygen en archivos .Rd.

Roxigen permite utilizar la descripción de los parámetros de otras funciones usando @inheritParams source function. Esta etiqueta traerá toda la documentación de los parámetros que no están documentados en la función actual, pero que están documentados en la función fuente. La fuente puede ser una función en el paquete actual, vía @inheritParams function, u otro paquete, vía @inheritParams package::function.

A diferencia de las funciones que son documentadas directamente, para los objetos en data/, se debe crear un archivo y guardarlo en el directorio R/.

3.2.4. Viñetas

EXPLICAR UN POCO QUE ES UNA VINETA

Para crear la viñeta, se utiliza usethis::use_vignette("my-vignette"). La misma crea un directorio vignettes/, agrega las dependencias necesarias a DESCRIPTION y redacta la viñeta. Las tres componentes fundamentales de la misma son las siguientes:

- El bloque inicial de metadatos, que contiene la siguiente información:

```
---
title: "Vignette Title"
output: rmarkdown::html_vignette
vignette: >
  %\VignetteIndexEntry{Vignette Title}
```

```
  %\VignetteEngine{knitr::rmarkdown}
  \usepackage[utf8]{inputenc}
---
```

- Markdown para formatear texto.
- Knitr para interpretar texto, código y resultados.

3.2.5. Pruebas del flujo de trabajo

Las pruebas resultan fundamentales en el desarrollo de paquetes, asegura que el código haga lo que realmente se desea. Existen pruebas informales como aquellas realizadas con la función load_all. Sin embargo, estas pruebas interactivas pueden convertirse en scripts reproducibles, los cuales resultan superiores debido a que:

- Se indica explícitamente cómo debería comportarse el código, provocando que los errores solucionados no vuelvan a ocurrir.
- El código que es fácil de probar generalmente está mejor diseñado, reduce la duplicación en el código. Como resultado, las funciones serán más fáciles de probar, comprender y trabajar.
- Si toda la funcionalidad del paquete tiene una prueba asociada, se pueden hacer grandes cambios sin preocuparse por generar errores.

Para probar el flujo de trabajo se utiliza la función usethis::usetestthat(). Esta crea un directorio tests/testthat, agrega testthat al campo Suggests en el archivo DESCRIPTION y además, crea un archivo tests/testthat.R que ejecuta todas las pruebas cuando R CMD check se ejecuta.

Las pruebas se organizan jerárquicamente: las expectativas se agrupan en pruebas que se organizan en archivos :

- Una expectativa describe el resultado esperado de un cálculo.
- Una prueba agrupa múltiples expectativas para probar la salida de una función simple, un rango de posibilidades para un solo parámetro de una función más complicada o una funcionalidad estrechamente relacionada de varias funciones.
- Un archivo agrupa múltiples pruebas relacionadas. Los archivos reciben un nombre legible para humanos con context().

Se utiliza la funcion test(), para ejecutar todas las pruebas creadas a la vez. Las pruebas también se ejecutan cada vez que se utiliza la función check() descripta a continuación.

3.2.6. Compilación e instalación

Mediante la función load_all se fue simulando el proceso de construcción, instalación y conexión del paquete, con el fin de ir probando las funciones creadas. Sin embargo, R CMD check ejecutado en el shell o check(), es el estándar de oro para verificar que un paquete R esté en pleno funcionamiento. La misma 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. Se aconseja realizar verificaciones completas de que todo funciona a medida que se van incorporando funciones ya que si se incorporan muchas y luego se verifican será dificil identificar y resolver los problemas. Una vez que las verificaciones completas no encuentran errores, advertencias o notas, se ejecuta la función install(), con el objetivo de instalar el paquete en la biblioteca.

3.2.7. 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 se deben seguir las siguientes instrucciones:

```
library(devtools)
install_github("jangelini/geneticae")
```

(Ideas de: https://mastering-shiny.org/dependency-tracking.html)

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 requiere de una instalación en el ordenador del usuario, simplemente se accede a través de un navegador. Por lo que es posible utilizar una aplicación web desde 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 R que te permite crear aplicaciones web interactivas, permitiendo exhibir un trabajo de R a través de un navegador web para que cualquiera pueda usarlo.

Crear aplicaciones web puede resultar difícil para la mayoría de los usuarios de R debido a que se necesita un conocimiento profundo de las tecnologías web como HTML, CSS y JavaScript; y además hacer aplicaciones interactivas complejas requiere un análisis cuidadoso de los flujos de interacción para asegurarse de que cuando una entrada cambie, solo se actualicen las salidas relacionadas. Sin embargo, shiny hace que sea mucho más fácil para el programador R crear aplicaciones web al proporcionar un conjunto de funciones de interfaz de usuario (UI para abreviar) que generan el HTML, CSS y JavaScript que necesita para tareas comunes. Esto significa que no se necesita conocer los detalles de HTML / CSS / JS.

Shiny fomenta la separación del código que genera tu interfaz de usuario (el *front-end*) del código que impulsa el comportamiento de tu aplicación (el *backend*. Por lo tanto, Los dos componentes clave de cada una Shiny APP son:

- 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 esquema interno de la aplicación puede observarse en la Figura 3.7.

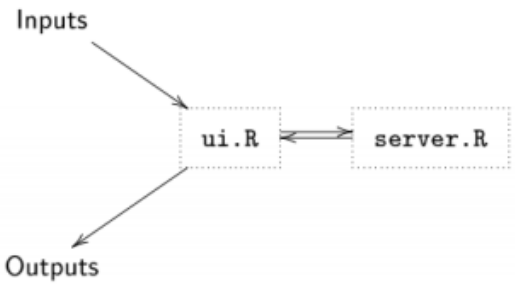


Figura 3.7: Esquema interno de la aplicación.

La programación reactiva es un estilo de programación que enfatiza valores que cambian con el tiempo, y cálculos y acciones que dependen de esos valores. Esto es importante para las aplicaciones Shiny porque son interactivas: los usuarios cambian los controles de entrada (arrastrando controles deslizantes, escribiendo en cuadros de texto y marcando casillas de verificación), lo que hace que la lógica se ejecute en el servidor (leer CSV, subconjunto de datos y modelos de ajuste) que finalmente resultan en actualización de salidas (parcelas de respuesta, actualización de tablas).

Para que las aplicaciones Shiny sean útiles, necesitamos dos cosas:

Las expresiones y las salidas deben actualizarse siempre que cambie uno de sus valores de entrada. Esto asegura que la entrada y la salida permanezcan sincronizadas.

Las expresiones y las salidas deben actualizarse solo cuando una de sus entradas cambia. Esto garantiza que las aplicaciones respondan rápidamente a la entrada del usuario, haciendo la cantidad mínima.

Es relativamente fácil satisfacer una de las dos condiciones, pero es mucho más difícil satisfacer ambas. Para ver por qué y para ver cómo podríamos atacar el problema básico con otros estilos de programación, utilizaremos un ejemplo muy simple, eliminando la complejidad adicional de una aplicación web y enfocándonos en el código subyacente.

3.3.1. Flujo de trabajo

El esta sección se motrará como mejorar dos flujos de trabajo de Shiny importantes: El ciclo de desarrollo básico de crear aplicaciones, realizar cambios y experimentar con los resultados; y la Depuración, el arte y el oficio de descubrir qué salió mal con su código y las soluciones de lluvia de ideas para solucionarlo.

1. Flujo de trabajo de desarrollo

El objetivo de optimizar el flujo de trabajo de desarrollo es reducir el tiempo entre hacer un cambio y ver el resultado. Cuanto más rápido se pueda iterar, más rápido se podrá experimentar y más rápido podrá obtener su Shiny. Aquí hay dos flujos de trabajo principales para optimizar: crear la aplicación por primera vez y acelerar el ciclo iterativo de ajustar el código y probar los resultados.

Creación de la Shiny APP

Para poder crear una shiny APP se debe tener instado R, RStudio ya que tiene algunas características agradables específicamente para autoría, depuración y desplegando aplicaciones estas aplicaciones, y lo siguientes paquetes:

```
install.packages(c(
  "ggforce", "shiny", "shinythemes", "tidyverse", "vroom"
))
```

Para crear una Shiny APP, lo más simple es crear un nuevo directorio para su aplicación y crear un sólo archivo llamado app.R en él. Este archivo se usará para indicarle a Shiny cómo debería verse la aplicación y cómo debería comportarse.

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

Por lo tanto, en el archivo app.R se realizan las siguientes tareas:

- Carga el paquete shiny:

  `library(shiny)`

- Define la interfaz de usuario, la página web HTML con la que los usuarios interactúan.
- Especifica el comportamiento de nuestra aplicación definiendo la función server.
- Se ejecuta shinyApp(ui, server)para construir e iniciar una aplicación Shiny desde la interfaz de usuario y el servidor.

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.

En todo tipo de programación, es una mala práctica tener código duplicado; puede ser un desperdicio computacional y, lo que es más importante, aumenta la dificultad de mantener o depurar el código.

En la secuencia de comandos R tradicional, utilizamos dos técnicas para lidiar con el código duplicado: capturamos el valor usando una variable o capturamos el cálculo con una función. Desafortunadamente, ninguno de estos enfoques funciona aquí y se necesita un nuevo mecanismo: expresiones reactivas . Una expresión reactiva tiene una diferencia importante con una variable: sólo se ejecuta la primera vez que se llama y luego almacena en caché el resultado de la misma hasta que necesite actualizarse.

Ver los cambios

Al crear o modificar la aplicacion, se la ejecuta para poder ver los cambios realizados, por lo que el dominio de flujo de trabajo de desarrollo es especialmente importante. La primera forma de reducir la velocidad de iteración es evitar hacer clic en el botón .Ejecutar aplicaciónz, en su lugar, aprender el método abreviado de teclado Cmd/Ctrl+ Shift+ Enter. Esto le brinda el siguiente flujo de trabajo de desarrollo:

1. Escribe un código.
2. Inicie la aplicación con Cmd/Ctrl+ Shift+ Enter.
3. Experimentar interactivamente con la aplicación.
4. Cierra la aplicación.
5. Ir a 1.

Otra forma de reducir aún más la velocidad de iteración es activar la recarga automática (options(shiny.autoreload = TRUE)) y luego ejecutar la aplicación en un trabajo en segundo plano, como se describe en https://github.com/sol-eng/background-jobs/tree/master/shiny-trabajo. Con este flujo de trabajo tan pronto como guarde un archivo, su aplicación se reiniciará: no es necesario cerrarla y reiniciarla. Esto conduce a un flujo de trabajo aún más rápido:

1. Escriba un código y presione Cmd/Ctrl+ Spara guardar en el archivo.
2. Experimentar interactivamente.
3. Ir a 1.

La principal desventaja de esta técnica es que debido a que la aplicación se ejecuta en un proceso separado, es considerablemente más difícil de depurar.

De manera predeterminada, cuando ejecuta la aplicación, aparecerá en una ventana emergente. Sin embargo ,existen otras dos opciones que puede elegir del menú desplegable *Run App*

1. La ejecución en el panel del visor es útil para aplicaciones más pequeñas porque puede verla al mismo tiempo que ejecuta el código de la aplicación.

2. Ejecutar en un navegador externo es útil para aplicaciones más grandes, o si desea verificar que su aplicación se ve exactamente de la manera que espera en el contexto que la mayoría de los usuarios la verán.

2. Depuración

Entre los problemas que pueden surgir al crear una Shiny app se encuentran los siguientes:

- Error inesperado. Este es el caso más fácil, porque obtendrá un rastreo que le permitirá averiguar exactamente de dónde proviene el error. Una vez que haya identificado el problema, deberá probar sistemáticamente su suposición hasta que encuentre una diferencia entre sus expectativas y lo que realmente está sucediendo. El depurador interactivo es una herramienta poderosa para este proceso.

- No obtiene ningún error, pero un valor es incorrecto. Aquí, generalmente es mejor transformar esto en el primer problema utilizando stop()para arrojar un error cuando se produce el valor incorrecto.

- Todos los valores son correctos, pero no se actualizan cuando espera. Este es el problema más desafiante porque es exclusivo de Shiny, por lo que no puede aprovechar sus habilidades de depuración de R.

Una vez localizado la fuente del error, la herramienta más poderosa es el depurador interactivo. El depurador detiene la ejecución y le brinda una consola R interactiva donde puede ejecutar cualquier código para descubrir qué salió mal. Hay dos formas de iniciar el depurador:

- Agregar una llamada a la función browser() en código fuente. Esta es la forma estándar de R de iniciar el depurador interactivo, y funcionará sin embargo, se está ejecutando brillante.

- Agregar un punto de interrupción RStudio haciendo clic a la izquierda del número de línea. Puede eliminar el punto de interrupción haciendo clic en el círculo rojo.

La ventaja de los puntos de interrupción es que no son código, por lo que nunca tendrá que preocuparse por registrarlos accidentalmente en su sistema de control de versiones.

3.3.2. 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*

El paquete *geneticae* ofrece funciones para el análisis de datos de etapas avanzadas de programas de mejoramiento, donde se evalúan pocos genotipos. Para instalarlo se deben utilizar las siguientes sentencias:

```
install.packages("devtools")
library(devtools)
install_github("jangelini/geneticae")
```

Una vez que se instala el paquete geneticae, se debe cargar en la sesion de R:

```
library(geneticae)
```

Se puede obtener información detallada sobre las funciones del paquete geneticae de los archivos de ayuda mediante *help (package = "geneticae")*. La ayuda para una función, por ejemplo, imputación, en una sesión R se puede obtener usando *?imputation* o *help(imputation)*.

4.1.1. Conjuntos de datos incluidos

El paquete geneticae proporciona dos conjuntos de datos para ilustrar la metodología incluida para analizar los datos MET.

- yan.winterwheat dataset: rendimiento de 18 variedades de trigo de invierno cultivadas en nueve ambientes en Ontario en 1993. No hay réplicas disponibles en los datos. Este conjunto de datos se obtuvo del paquete agridat.

  ```
  library(geneticae)
  data(yan.winterwheat)
  ```

```
dat_yan <- yan.winterwheat
head(dat_yan)
```

```
##   gen  env yield
## 1 Ann BH93 4.460
## 2 Ari BH93 4.417
## 3 Aug BH93 4.669
## 4 Cas BH93 4.732
## 5 Del BH93 4.390
## 6 Dia BH93 5.178
```

- plrv dataset: rendimiento, peso de planta y parcela de 28 clones de la población del virus del enrollamiento de la papa (PLRV) evaluada en seis entornos. Las réplicas están disponibles en los datos. Este conjunto de datos se obtuvo del paquete agricolae.

```
data(plrv)
dat_rep <- plrv
head(dat_rep)
```

```
##   Genotype Locality Rep WeightPlant WeightPlot    Yield
## 1   102.18     Ayac   1   0.5100000       5.10 18.88889
## 2   104.22     Ayac   1   0.3450000       2.76 12.77778
## 3   121.31     Ayac   1   0.5425000       4.34 20.09259
## 4   141.28     Ayac   1   0.9888889       8.90 36.62551
## 5   157.26     Ayac   1   0.6250000       5.00 23.14815
## 6    163.9     Ayac   1   0.5120000       2.56 18.96296
```

4.1.2. Funciones incluidas

Modelo de regresión por sitio

Para ejecutar la función *GGEmodel*, se debe proporcionar un conjunto de datos con genotipos, ambientes, repeticiones (si hay disponibles), el fenotipo observado y los nombres de las variables en el archivo de entrada. Además, se debe indicar el método de centrado, escala y SVD.

Cuando no hay repeticiones disponibles en el conjunto de datos, como es el caso del conjunto de datos yan.winterwheat, el modelo GGE se indica de la siguiente manera:

```
GGE1 <- GGEmodel(dat_yan, genotype = "gen", environment = "env",
response = "yield", centering = "tester")
```

Sin embargo, en el caso de que haya repeticiones disponibles, como el conjunto de datos plrv, se indica de la siguiente manera:

```
GGE1_rep <- GGEmodel(dat_rep, genotype = "Genotype", environment = "Locality",
response = "Yield", rep = "Rep", centering = "tester")
```

La salida de la función GGEmodel es una lista con los siguientes elementos:

- coordgenotype: trazado de coordenadas para genotipos de todos los componentes.
- coordenviroment: trazado de coordenadas para entornos de todos los componentes.
- valores propios: vector de valores propios de cada componente.
- vartotal: varianza general.
- varexpl: porcentaje de varianza explicado por cada componente.
- labelgen: nombres de genotipo.
- labelenv: nombres de entorno.
- ejes: etiquetas de eje.
- Datos: datos de entrada escalados y centrados.
- centrado: nombre del método de centrado.
- escala: nombre del método de escala.
- SVP: nombre del método SVP.

Por ejemplo, para el conjunto de datos yan.winterwheat:

```
names(GGE1)
```

```
##  [1] "coordgenotype"   "coordenviroment" "eigenvalues"
##  [4] "vartotal"        "varexpl"         "labelgen"
##  [7] "labelenv"        "labelaxes"       "Data"
## [10] "centering"       "scaling"         "SVP"
```

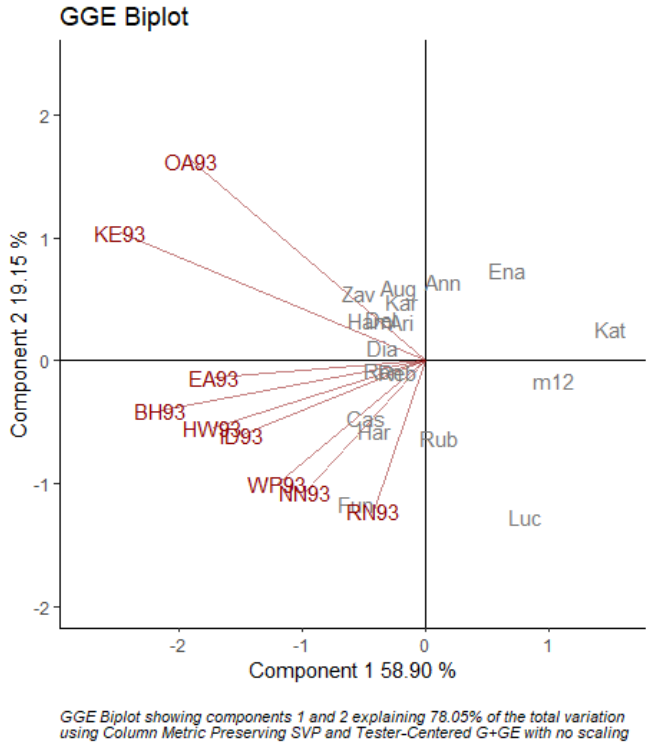
Biplot GGE

Para ejecutar la función *GGEPlot*, se requiere un objeto de la clase *GGEmodel*. La salida es un biplot construido a través de los componentes principales generados por *GGEmodel*.

Los diferentes biplots que se pueden obtener usando la función *GGEPlot* se muestran usando el conjunto de datos yan.winterwheat.

- Biplot básico, se obtiene indicando type = "Biplot".

```
GGEPlot(GGE1, type = "Biplot")
```



GGE Biplot showing components 1 and 2 explaining 78.05% of the total variation using Column Metric Preserving SVP and Tester-Centered G+GE with no scaling

Figura 4.1: Biplot básico obtenido de la función *GGEPlot*

- Biplot básico, se obtiene indicando type = "Biplot".
- Ranking de los cultivares en función de su rendimiento en cualquier ambiente dado se obtiene indicando type="Selected Environmentz el nombre del ambiente para examinarse selectedE = .ºA93".

```
GGEPlot(GGE1, type = "Selected Environment", selectedE = "OA93")
```

- Ranking de los ambientes en función del rendimiento relativo de cualquier cultivar determinado se obtiene indicando type="Selected Genotypez el nombre del ambiente para examinarse selectedG = "Kat".

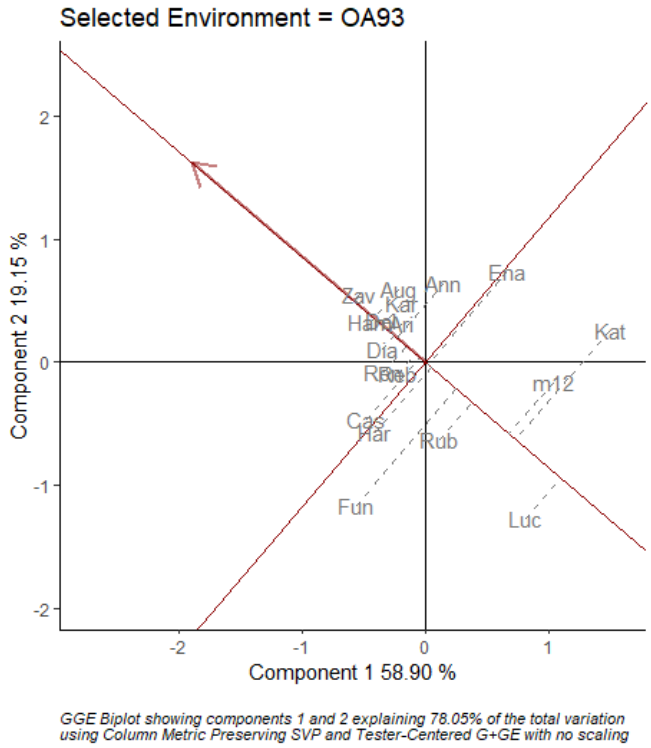


Figura 4.2: Ranking de cultivares para un ambiente determinado obtenido de la función *GGEPlot*

```
GGEPlot(GGE1, type = "Selected Genotype", selectedG = "Kat")
```

- Relación entre ambientes, se obtiene indicando type=Relationship Among Environments".

```
GGEPlot(GGE1, type = "Relationship Among Environments")
```

- Comparación entre dos genotipos se obtiene indicando type = Çomparison of Genotypez los genotipos para comparar selectedG1 = "Katçon selectedG2 = Ças".

```
GGEPlot(GGE1, type = "Comparison of Genotype", selectedG1 = "Kat", selectedG2 =
```

- Identificación del mejor cultivar en cada ambiente indicando type = "Which Won Where/What".

```
GGEPlot(GGE1, type = "Which Won Where/What")
```

- Evaluación de los ambientes basados tanto en la capacidad de discriminación como en la representatividad indicando type = "Which Won Where/What".

```
GGEPlot(GGE1, type = "Discrimination vs. representativeness")
```

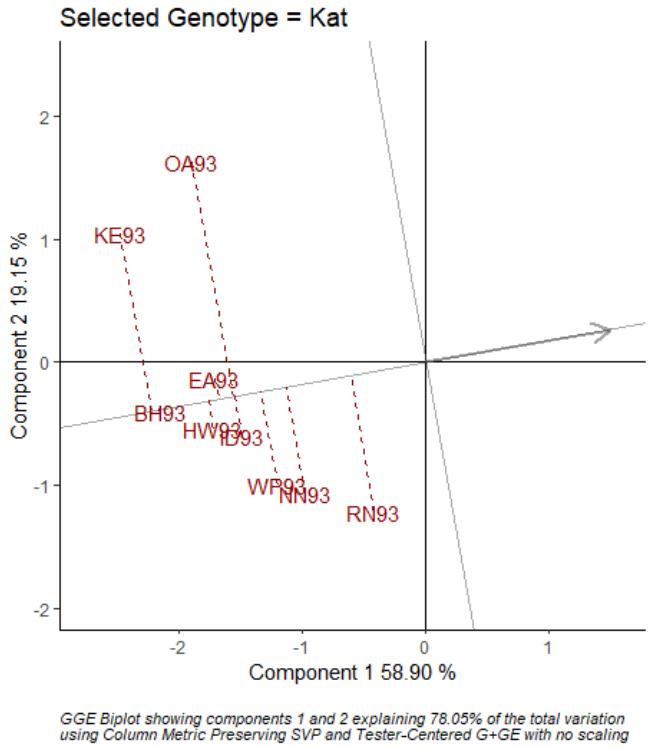


Figura 4.3: Ranking de ambientes para cultivar determinado obtenido de la función *GGEPlot*

- Clasificación de ambientes con respecto al ambiente ideal, indicando type=Ranking Environments".

```
GGEPlot(GGE1, type = "Ranking Environments")
```

- Clasificación de genotipos con respecto al genotipo ideal, indicando type = Ranking Genotypes".

```
GGEPlot(GGE1, type = "Ranking Genotypes")
```

- Evaluación de los cultivares con base en el rendimiento promedio y la estabilidad, indicando type = "Mean vs. Stability".

```
GGEPlot(GGE1, type = "Mean vs. Stability")
```

Classic AMMI model

Para ejecutar la función rAMMI, como en la función GGEmodel, se debe proporcionar un conjunto de datos con genotipo, entorno, repeticiones (si las hay) y la variable de respuesta. Se debe indicar el nombre de las columnas que contienen genotipo, entorno, repeticiones y la respuesta. La salida es un biplot.

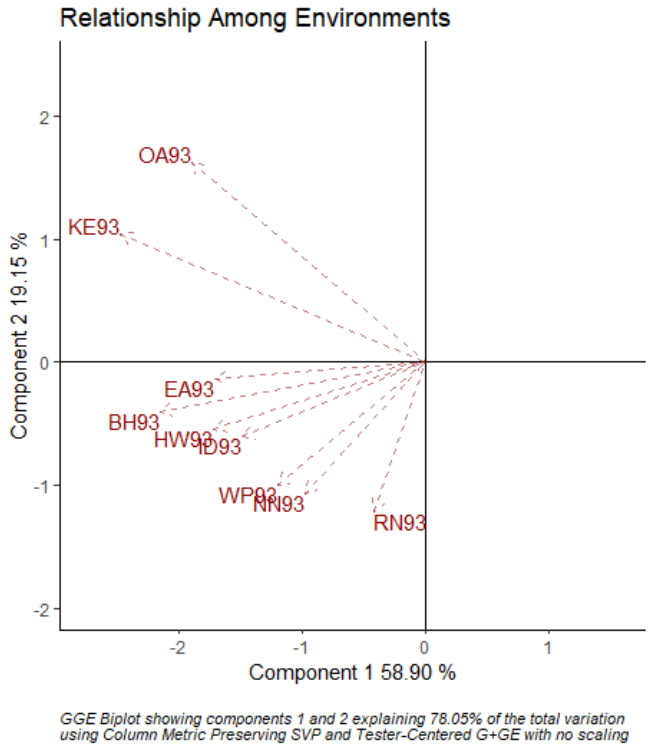


Figura 4.4: Relación entre ambientes obtenido de la función *GGEPlot*

A continuación se muestra el biplot GE obtenido del modelo AMMI clásico obtenido con el conjunto de datos yan.winterwheat.

```
rAMMI(dat_yan, genotype = "gen", environment = "env", response = "yield", type = "A
```

Robust AMMI model

Los biplots de los cinco modelos AMMI robustos propuestos por Rodrigues et al. (2015), usando el conjunto de datos yan.winterwheat se muestran a continuación.

- type = rAMMI"

  ```
  rAMMI(dat_yan, genotype = "gen", environment = "env", response = "yield", type
  ```

- type = "hAMMI"

  ```
  rAMMI(dat_yan, genotype = "gen", environment = "env", response = "yield", type
  ```

- type = "gAMMI"

  ```
  rAMMI(dat_yan, genotype = "gen", environment = "env", response = "yield", type
  ```

- type = "lAMMI"

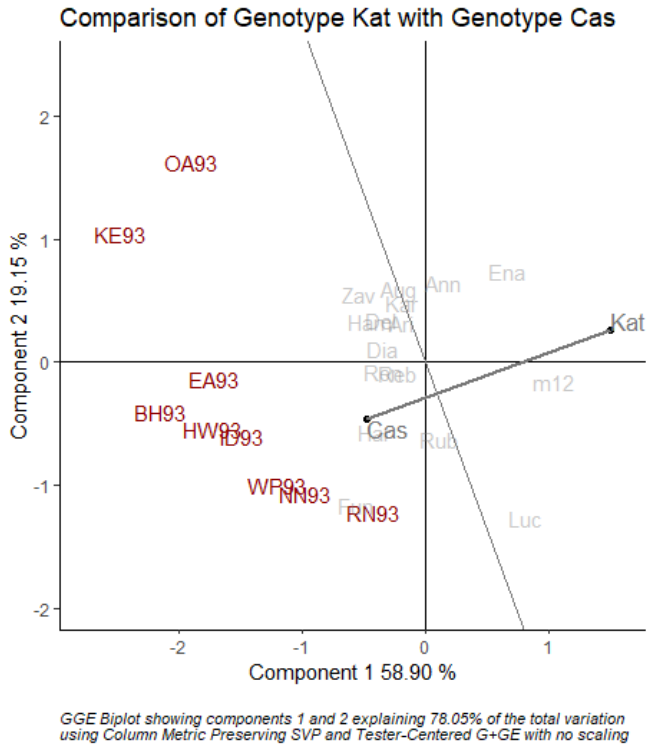


Figura 4.5: Comparación entre dos genotipos obtenido de la función *GGEPlot*

```
rAMMI(dat_yan, genotype = "gen", environment = "env", response = "yield", type
```

- type = "ppAMMI"

```
rAMMI(dat_yan, genotype = "gen", environment = "env", response = "yield", type
```

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

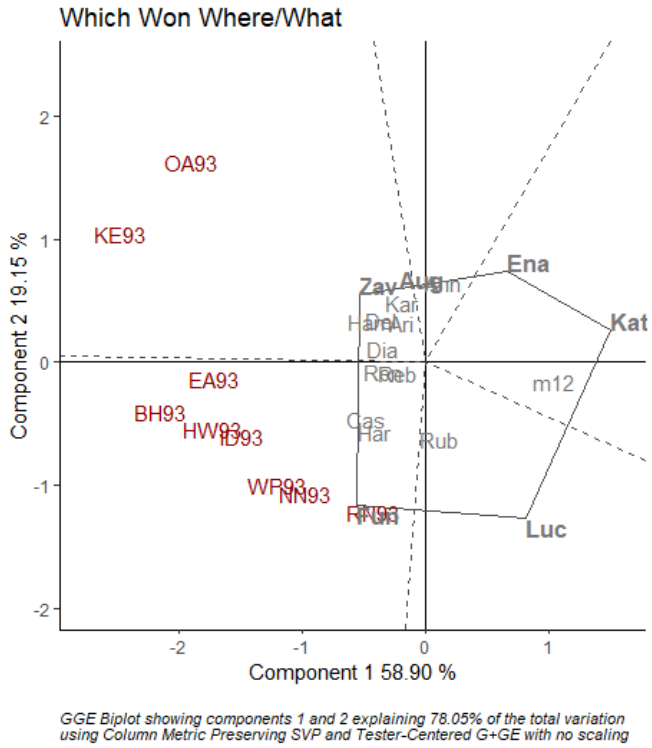


Figura 4.6: Identificación del mejor cultivar en cada ambiente a partir de la función *GGEPlot*

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).

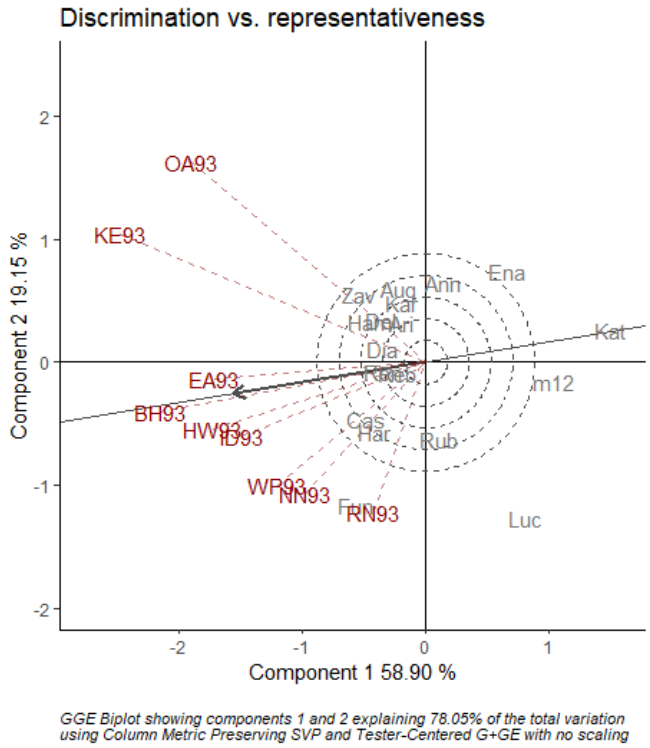


Figura 4.7: Evaluación de los ambientes basados tanto en la capacidad de discriminación y representatividad a partir de la función *GGEPlot*

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.

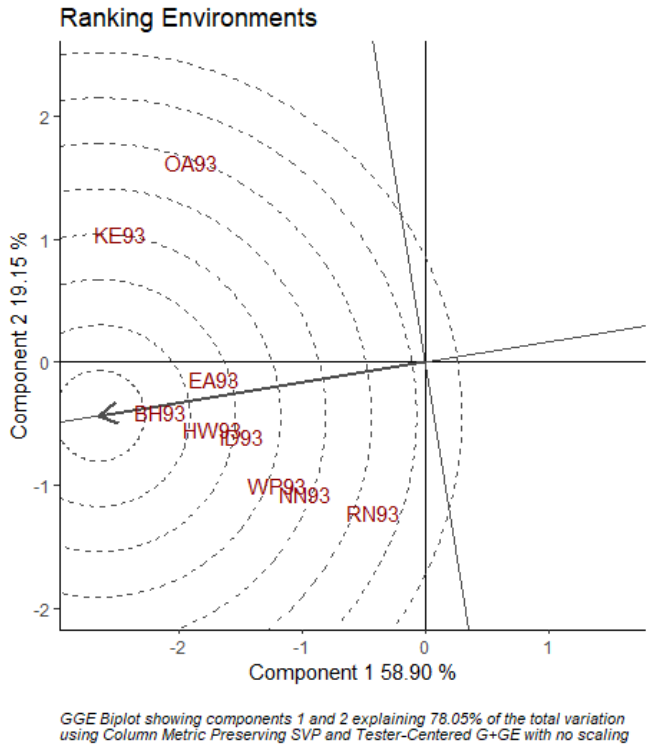


Figura 4.8: Clasificación de ambientes con respecto al ambiente ideal a partir de la función *GGEPlot*

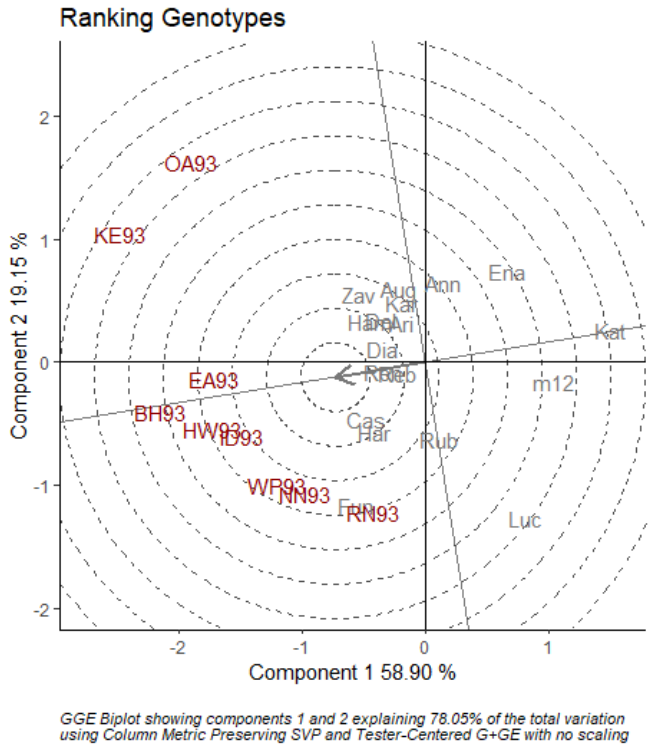


Figura 4.9: Clasificación de genotipos con respecto al genotipo ideal a partir de la función *GGEPlot*

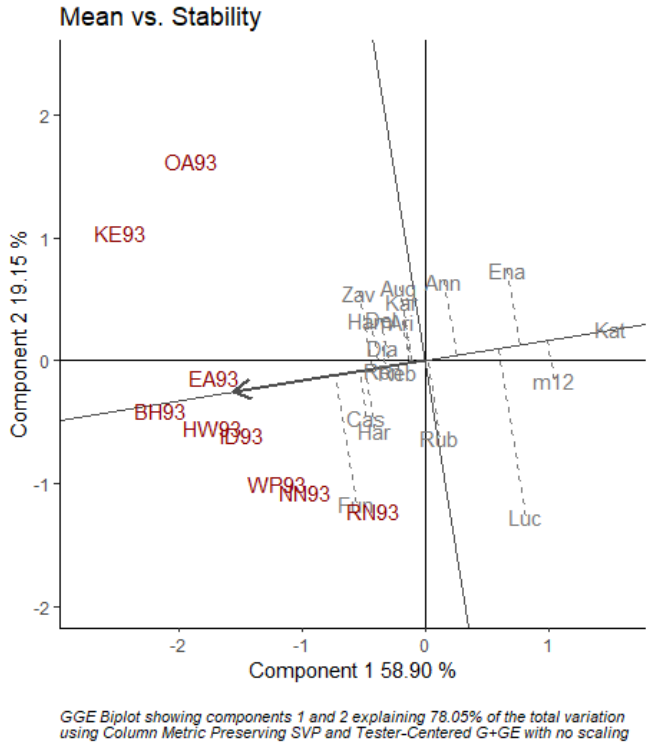


Figura 4.10: Evaluación de los cultivares con base en el rendimiento promedio y la estabilidad a partir de la función *GGEPlot*

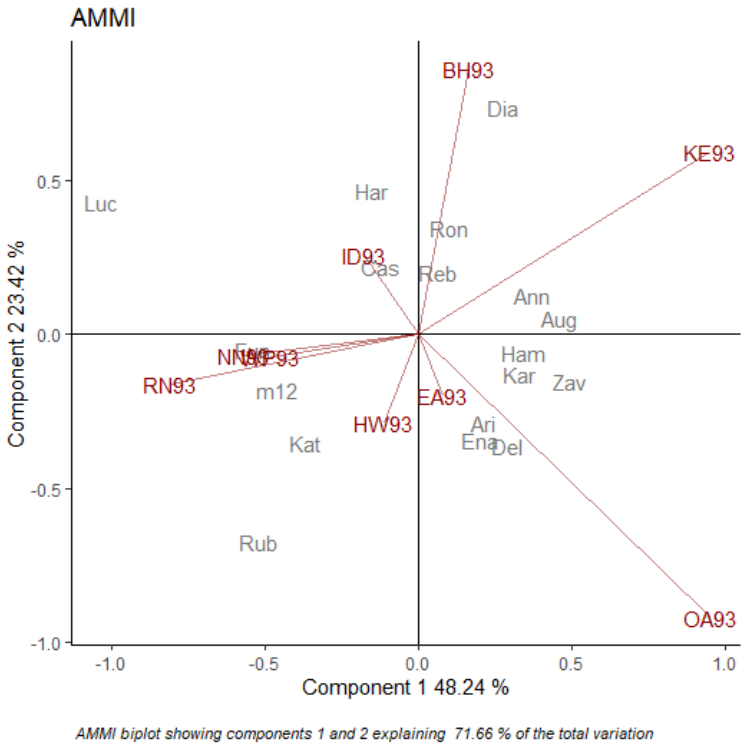


Figura 4.11: Biplot GE obtenido del modelo clasico AMMI

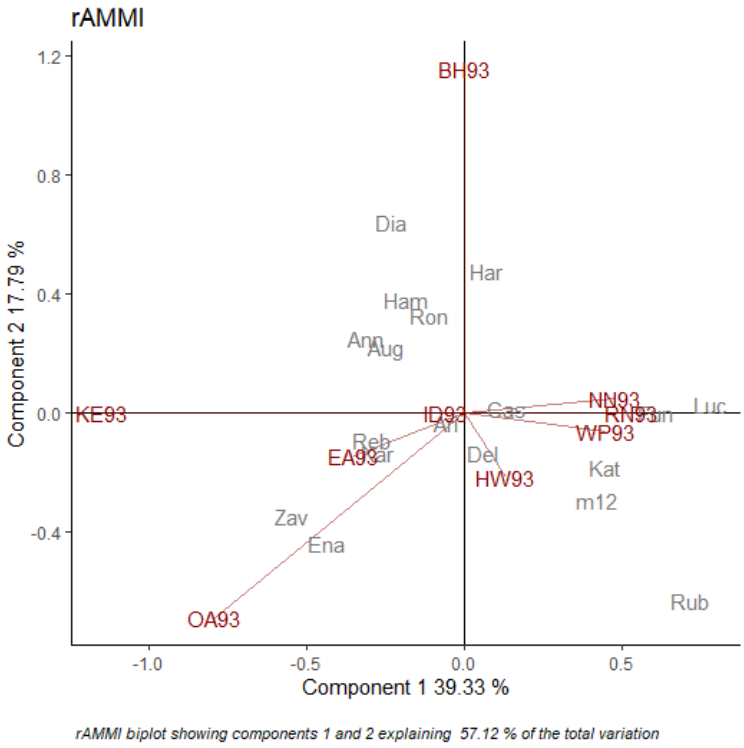


Figura 4.12: Biplot GE obtenido del modelo robusto rAMMI

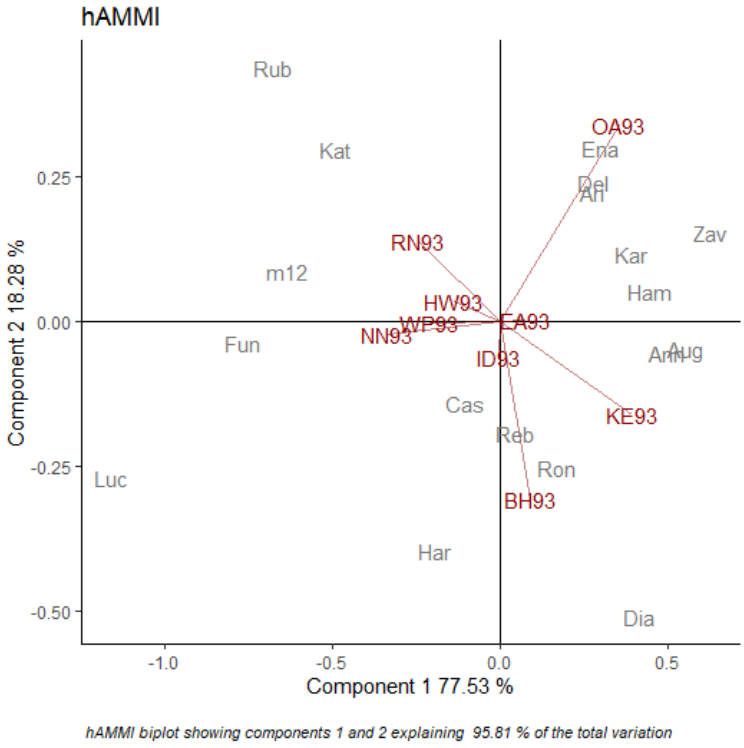


Figura 4.13: Biplot GE obtenido del modelo robusto hAMMI

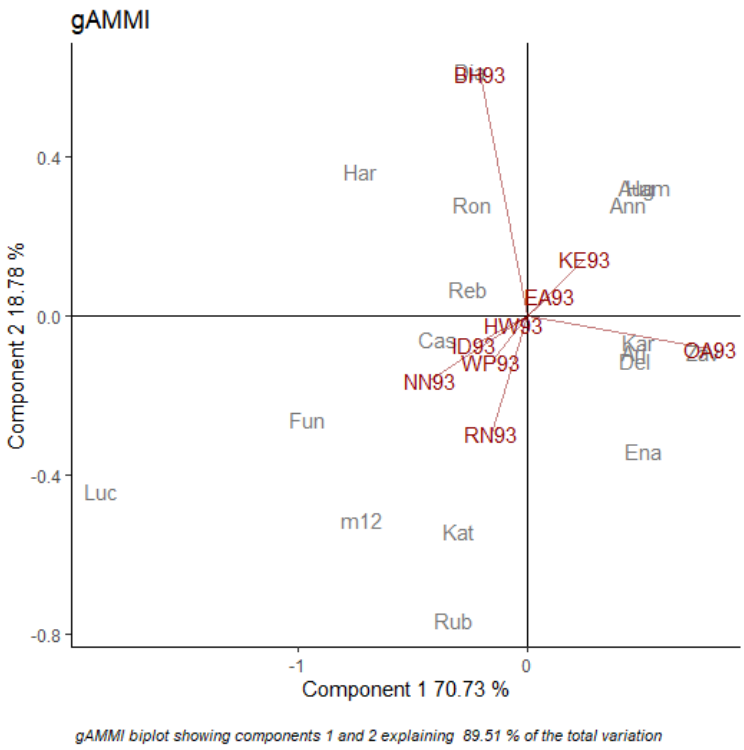


Figura 4.14: Biplot GE obtenido del modelo robusto hAMMI

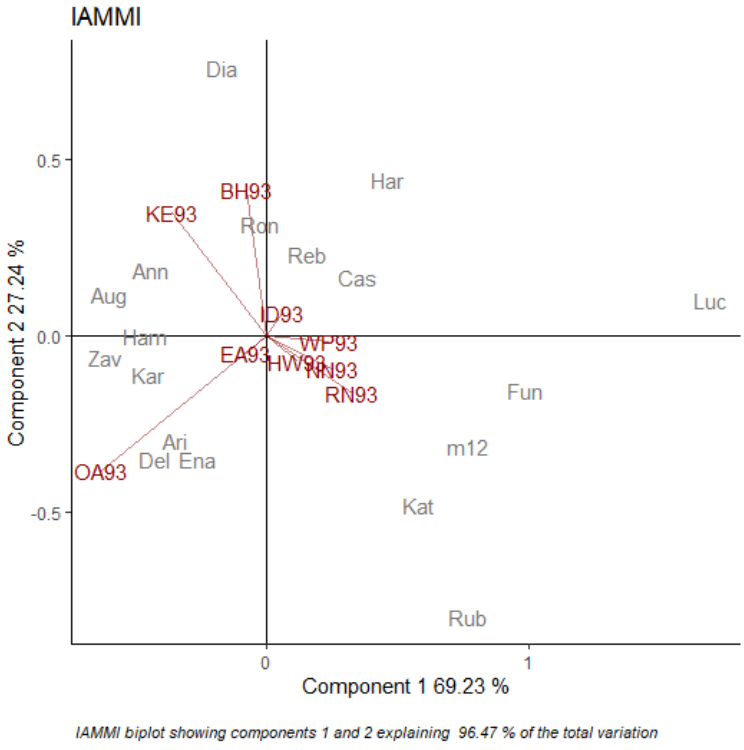


Figura 4.15: Biplot GE obtenido del modelo robusto hAMMI

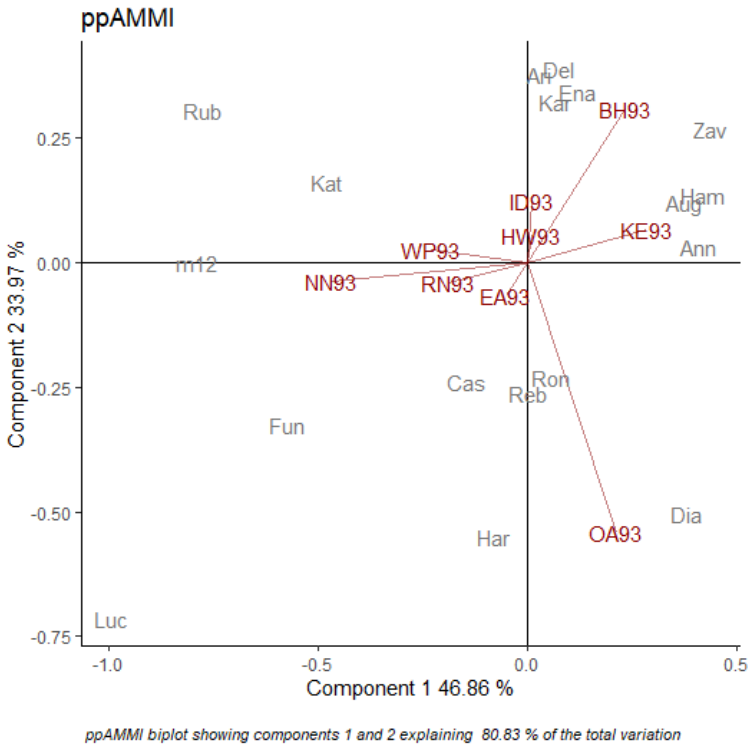


Figura 4.16: Biplot GE obtenido del modelo robusto hAMMI

Apéndice A

Hoja de referencia Shiny

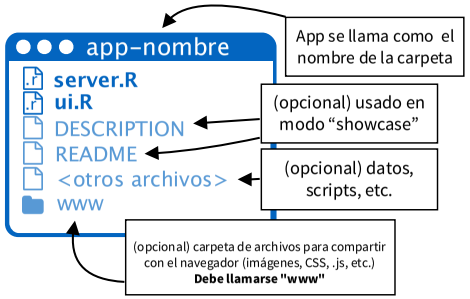
Shiny Hoja de referencia

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:

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

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

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

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

E 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.

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

server.R

```
# carga paquetes, scripts, datos

A shinyServer(function(input, output) {B

  # crea variables específicos para usuario

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

C   output$gráfica <- renderPlot({D
      x <- mtcars[ , input$x]F
E     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.

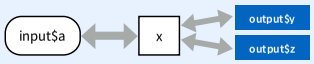
Corre a menudo - código puesto dentro de una función *render**, *reactive*, o *observe* correrá muchas veces. Úsalo solo para código que el servidor necesita para reconstruir un componente UI después de que un widget cambia.

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

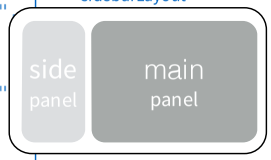
5. ui.R Una descripción de la interfaz (UI) de tu app, la página web que muestra tu app.

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

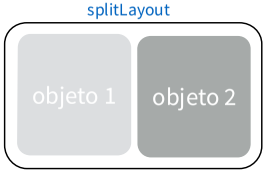
B Construye el plano para tu UI. **sidebarLayout** da una composición estándar cuando se usa con **sidebarPanel** y **mainPanel**. **splitLayout**, **flowLayout**, e **inputLayout** dividen la página en regiones equidistantes. **fluidRow** y **column** trabajan juntos para crear planos basados en rejillas,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

sidebarLayout



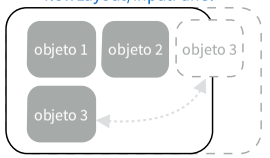

```
shinyUI(fluidPage(
  sidebarLayout(
    sidebarPanel(…),
    mainPanel(…)
  )
))
```

splitLayout



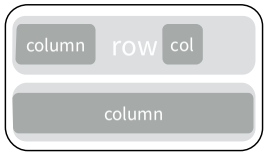

```
shinyUI(fluidPage(
  splitLayout(
    numericInput(…),
    selectInput(…)
  )
))
```

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

Bibliografía

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

H. G. Gauch y R. W. Zobel. Identifying mega-environments and targeting genotypes. *Crop Science*, 37:311—-326, 1997.

W. Yan, L. A. Hunt, Q. Sheng, y Z. Szlavnics. Cultivar evaluation and mega-environment investigation based on the GGE biplot. *Crop Science*, 40:597—-605, 2000.