UNIVERSIDAD DE SONORA

DIVISIÓN DE CIENCIAS EXACTAS Y NATURALES
DEPARTAMENTO DE MATEMÁTICAS
LIC. EN CIENCIAS DE LA COMPUTACIÓN

RADIOGRAFIAR MÉTODOS DE CAJA NEGRA USANDO TÉCNICAS ESTADÍSTICAS

T E S I S

QUE PARA OBTENER EL TÍTULO DE

LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A

BRENDA JESÚS RODRÍGUEZ ALCÁNTAR

Director de Tesis

DR. JOHAN JOZEF LODE VAN HOREBEEK

Hermosillo, Sonora Febrero de 2007

Universidad de Sonora

Repositorio Institucional UNISON

Radiografiar Métodos de Caja Negra usando Técnicas Estadísticas

por

Brenda Jesús Rodríguez Alcántar

Nombre del Autor .....

Brenda Jesús Rodríguez Alcántar

Director de tesis .....

Dr. Johan Jozef Lode Van Horebeek

Aceptado por .....

M. C. Sonia Guadalupe Sosa León
Revisor de tesis

Aceptado por .....

M. C. Gudelia Figueroa Preciado
Revisor de tesis

Aceptado por .....

M. C. Pedro Flores Pérez
Revisor de tesis

Radiografiar Métodos de Caja Negra usando Técnicas Estadísticas

por

Brenda Jesús Rodríguez Alcántar

Resumen

El presente escrito es una tesis de licenciatura en ciencias de la computación, titulado *Radiografiar Métodos de Caja Negra usando Técnicas Estadísticas*. En este trabajo se expondrán dos mecanismos para la clasificación de datos: bosques al azar y redes neuronales artificiales, considerando ambos como una caja negra de la cual se desconoce su funcionamiento y es por medio de la implementación de un programa computacional, se pretende radiografiar el interior de la caja con la finalidad de ver que ocurre dentro de ella.

El área en la cual se trabaja es: Estadística computacional y Aprendizaje automático.

Se trabajó con el *Lenguaje de Programación Estadístico R*. Se escogió este lenguaje, debido a que es una herramienta muy conocida dentro del área de estadística, ya que proporciona varios y muy completos paquetes para el análisis de datos y además cuenta con la ventaja de ser un lenguaje multiplataforma y una distribución gratuita.

A mis padres y hermanas,
pero muy en especial a
Edelmira Rodríguez Alcántar,
por todo el apoyo brindado.

A Edel con todo mi corazón.

Agradecimientos

Muchas han sido las personas que de manera directa o indirecta me han ayudado en la realización de esta tesis. Quiero dejar constancia de todas ellas y agradecerles con sinceridad su participación.

Mi más sincero agradecimiento al Dr. Johan Jozef Lode Van Horebeek, Profesor-Investigador del Centro de Investigación en Matemáticas, de Guanajuato, Gto. por la oportunidad de trabajar bajo su coordinación. Así como al Centro de Investigación en Matemáticas por el apoyo económico e instalaciones proporcionados para la realización de este proyecto de tesis.

A los tres sinodales de tesis, los cuales con sus comentarios y sugerencias han mejorado considerablemente la calidad de este trabajo.

No podría dejar de agradecer a mi familia por el apoyo incondicional y a Edel Rafael Rodea Montero por la asesoria proporcionada.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

La clasificación es un proceso de extracción de conocimiento a partir de datos. Podemos decir que para identificar un objeto, dentro de un conjunto de varios objetos diferentes, es necesario primero conocer las características que los hacen desiguales entre ellos. Para ese fin, debemos de aprender a diferenciar esos atributos y después poder reconocer las particularidades de un nuevo objeto.

Partiendo del método de enseñanza tradicional, donde un profesor indica y corrige los errores de sus alumno hasta que estos aprendes la lección, observamos que para realizar el proceso de enseñanza, el maestro provee de ejemplos (o ejercicios) a sus alumnos, con el fin de que aprendan y posteriormente procede a evaluar las habilidades adquiridas por medio de nuevos ejercicios.

Lo anterior lo podemos definir como aprendizaje supervisado, ya que el profesor está supervisando las mejoras del alumno. Desde el punto de vista computacional este tipo de aprendizaje es bastante utilizado en el entrenamiento de algoritmos de clasificación de datos, por ejemplo *Redes Neuronales*, donde de igual manera se utilizan dos conjuntos de ejemplos (también denominados conjuntos de patrones), uno para enseñar a distinguir las diferencias entre los patrones que forman los datos y otro más para probar la eficiencia del entrenamiento, solo que bajo el contexto del área de *Inteligencia Artificial* a dichos conjuntos se les conoce como, archivo de

entrenamiento (con este nombre nos referimos al conjunto de patrones destinados para enseñar, en otras palabras para entrenar, a un algoritmo de clasificación) y archivo de prueba (con este nombre nos referimos al conjunto de patrones destinados a probar o medir la eficiencia del entrenamiento).

Siguiendo con la analógia del profesor y el alumno, en el archivo de entrenamiento se presentan ejemplos con los cuales se instruye al alumno y con el archivo de prueba se evalúa el nivel de aprendizaje del mismo.

Para realizar una clasificación de patrones, primero debemos de contar con los datos. Imaginemos que para obtener un conjunto de datos, introducimos un vector de variables de entrada $\boldsymbol{X}$ al interior de una caja, la cual regresa como respuesta una variable Y, ver Figura 1.1. En el interior de la caja ocurre "algo" para asociar el vector de entrada con la variable de salida. La pregunta es: ¿Qué sucede dentro de la caja que asocia las variables predictivas con las variables de respuesta?.

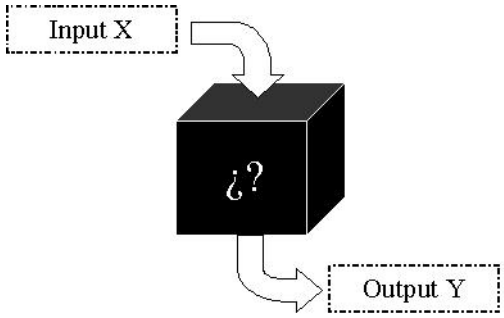


Figura 1.1: Caja Negra

Siguiendo a Leo Breiman [12] existen dos enfoques, sobre lo que sucede dentro de la caja.

En el primer enfoque se trata de entender que pasa en la caja donde la gente asume que los datos son generados por algún modelo, dicho enfoque se centra en el mecanismo subyacente y en la pregunta ¿Por qué ese modelo?, (ver Figura 1.2). El análisis comienza con asumir para el interior de la caja un modelo estocástico

(paramétrico) para los datos (que su origen es al azar, pero puede no parecerlo, dado que se presentan en forma regular). Los valores de los parámetros del modelo son estimados desde los datos y entonces el modelo es usado para dar información y/o hacer predicciones.

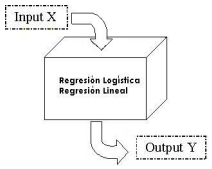


Figura 1.2: Modelo de datos.

Como ejemplo, tenemos un conjunto de caracteristícas $X = (x_1, x_2, \ldots, x_k)$, donde x_1 representa el nivel socioeconómico de una persona, x_2 indica su sexo, etc. y como respuesta la variable Y que toma valores si o no e indica si el individuo examinado presenta sobrepeso. De lo anterior surge la pregunta, ¿Es importante conocer el nivel socioeconómico de alguien para determinar si sufre sobrepeso?.

En el segundo enfoque se usan algoritmos en lugar de modelos, donde se asume el mecanismo de generación de datos como desconocido; se centra la atención en el poder predictivo del clasificador, el enfoque pone énfasis en la pregunta ¿Cómo funciona el algoritmo? y considera el mecanismo de clasificación como una caja negra, (ver Figura 1.3). El análisis considera el interior de la caja como complejo y desconocido. Las aproximaciones del algoritmo sirven para encontrar una función *f(**X**)*, en otras palabras *f(**X**)* es un algoritmo que manipula las entradas denotadas por el vector $\boldsymbol{X}$ para predecir la variable de respuesta Y.

Como ejemplo, tenemos el poder clasificar un nuevo *e-mail* como *spam* o no *spam*, teniendo más de 100 variables predictoras. De lo anterior surge la pregunta, ¿Cómo podemos decidir bien si un nuevo *e-mail* es *spam*?.

Dentro de esta tesis nos concentraremos en el segundo enfoque, que abarca los

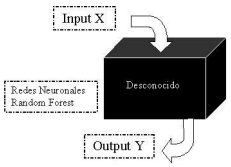


Figura 1.3: Modelos algorítmicos.

modelos algorítmicos, exponiendo dos mecanismos populares para la clasificación de datos, *Bosques al Azar* [11] y *Redes Neuronales Artificiales* [5, 10]. El método *Bosques al Azar* está inspirado en la representación de un bosque orgánico, el cual está formado por un conjunto de árboles naturales. La idea general es construir un clasificador de datos con base en la unión de muchos clasificadores sencillos que manipulan partes de la información de manera independiente entre ellos. El método *Redes Neuronales Artificiales* es un modelo computacional inspirado en el funcionamiento del cerebro humano con la intención de imitarlo. La idea es construir una función compleja a partir de funciones sencillas. En general, ambos métodos están cimentados en modelos naturales y tienen como objetivo crear un clasificador robusto, utilizando la fuerza conjunta de varios clasificadores sencillos. Viendo ambos clasificadores como un tipo de caja negra, se pretende en esta tesis, abrir dicha caja y observar que sucede dentro de ella; en otras palabras saber que ocurre con las variables de entrada para dar una respuesta y poder establecer cuales son realmente necesarias.

Para este fin se elaboró un programa computacional, bajo el *Lenguaje de Programación Estadístico R* [9], que es la implementación de *Context Sentive Regression Models* [4]. Dicho programa se ha realizado con el fin de que sea un paquete que se pueda adherir al lenguaje mencionado.

Como ya se indico en el resumen de la tesis, el área donde se trabaja es estadística computacional, por lo tanto, para poder realizar la presente tesis, se necesito de un conocimiento previo sobre el área de estadística, dado que los métodos utilizados tienen mucho peso estadístico.

1.1. Estructura del documento

El presente documento de tesis cuenta con la siguiente estructura:

En el Capítulo 2 se incluye la descripción de *Context Sensitive Regression Models*. El propósito de este método es determinar el impacto de cualquier variable de entrada en la variable de respuesta con base en su interacción con las demás variables de entrada.

Con este tema se realizará la parte principal del proposito original del proyecto, que como se recordara es radiografiar el interior de un modelo de tipo caja negra. En otras palabras lo que se busca a grandes rasgos es poder encontrar el mejor modelo *Context Sensitive Regression Models* que se ajusta a los datos utilizados, donde los datos utilizados serán resultados dados por alguno de los métodos de tipo caja negra que abarca el proyecto.

En el Capítulo 3 se aborda el desarrollo del software creado para la implentación de *Context Sensitive Regression Models*.

Dentro de este capítulo se incluyen los pasos que se siguieron para la implementación computacional del modelo *Context Sensitive Regression Models*. Cabe aclarar que el código no se incluye por cuestiones de espacio, sin embargo si se deseara consultar, la tesis incluye un disco compacto con el código fuente, así como también con todas las librerias que se necesitan y algunos conjuntos de datos a modo de ejemplos.

En el Capítulo 4 se describe detalladamente el modelo de tipo caja negra denominado *Bosques al Azar*.

El primer modelo de tipo caja negra que se ve es *Bosques al Azar* del cual se

explica su funcionamiento dentro de este capítulo, ya que éste modelo es bastante nuevo y poco conocido, se explica ampliamente. Con *Bosques al Azar* se realizaron varios clasificadores, los resultados obtenidos con él son utilizados por el método *Context Sensitive Regression Models*.

Si el interes del lector es solo conocer o entender mejor, en dado caso de que ya se tenga conocimiento previo sobre el tema, el método para realizar clasificación implementado en *Bosques al Azar*, se puede consultar este capítulo individualmente. Si se desea conocer más sobre el tema se pueden consultar [3, 6, 11].

En el Capítulo 5 se plantea y describe la parte del método nombrado *Redes Neuronales Artificiales* que se usará dentro de este trabajo.

El segundo modelo de tipo caja negra que se ve es *Redes Neuronales Artificiales*, dado que éste es un modelo muy conocido dentro del área de inteligencia artificial solo se abarca el caso de una arquitectura de red *unidireccional*, recordemos que *Redes Neuronales Artificiales* también es un clasificador por lo tanto, los resultados obtenidos con él son utilizados por el método *Context Sensitive Regression Models*.

Si el interes del lector es revisar el método de clasificación utilizado por *Redes Neuronales Artificiales* para una arquitectura de red *unidireccional*, se puede consultar este capítulo individualmente. Cabe señalar que se parte del hecho de que el lector tiene conocimientos previos de lo que son las redes neuronales, sin embargo, si no se contará con dicho conocimiento, se puede consultar este capítulo individualmente pero solo conocerá lo que es una red neuronal artificial para una arquitectura de red en particular. Si se desea conocer más sobre el tema de las redes neuronales artificiales se pueden consultar [1, 2, 5, 10].

En el Capítulo 6 se incluyen pruebas para el método *Context Sensitive Regression Models*. Se ve paso a paso el proceso que se sigue para poder ajustar un modelo

Context Sensitive Regression Models a un conjunto de datos.

El proceso se realiza para un conjunto de datos en especifico. Utilizando el conjunto de datos definido, se construye un clasificador con el primer modelo de tipo caja negra (*Bosques al Azar*), los resultados arrojados por el clasificador son utilizados para encontrar el mejor modelo de *Context Sensitive Regression Models* que se ajusta a los datos arrojados por el clasificador. El mismo proceso que se sigue con el primer modelo de tipo caja negra, se realiza también con el segundo método de tipo caja negra que se ve dentro de este proyecto (*Redes Neuronales Artificiales*), utilizando el conjunto de datos inicial.

Para ambos métodos de tipo caja negra, como ya se explico, se utiliza el mismo conjunto de datos para construir ambos clasificadores individualmente, el proposito de utilizar el mismo conjunto de datos para constriur los clasificadores y buscar el mejor modelo *Context Sensitive Regression Models* con los resultados individuales de cada método de tipo caja negra, es observar las diferencias y similitudes que se obtienen al utilizar ambos métodos de tipo caja negra. No se busca argumentar que uno es mejor que el otro, solo que son diferentes y que por la construcción misma de ambos métodos los resultados pueden llegar a ser diferentes.

En el Apéndice A se incluye una descripción de los conjuntos de datos utilizados, así como la dirección web de donde se obtuvieron o el proceso de generación utilizado, según sea el caso.

El Apéndice C presenta un rápido tutorial sobre el funcionamiento del lenguaje de programación **R**, con el fin de familiarizarse con él.

Capítulo 2

Context Sensitive Regression Models

Un *Context Sensitive Regression Model* (**Modelo CSR**) es un método que se basa en establecer el impacto que pueden generar las variables de entrada en una variable de respuesta. El propósito principal de este modelo es describir la manera en la que las variables de entrada afectan al resultado de una característica particular. Para ese fin, se usan representaciones gráficas y modelos de regresión lineal.

2.1. Introducción

El análisis de regresión lineal es una técnica estadística para modelar la relación entre la variable de entrada y la variable de respuesta:

$$Y = \beta_0 + \beta_1 x + \varepsilon. \tag{2.1}$$

A la ecuación (2.1) se le conoce como modelo de regresión lineal, donde β_0 y β_1 son parámetros desconocidos, x es la variable predictora, Y la variable de respuesta y ε es una variable aleatoria con esperanza cero. Puesto que sólo se tiene una variable predictora, se le denomina *modelo de regresión lineal simple*. La ecuación (2.1) equivale a $E(Y) = \beta_0 + \beta_1 x$.

Si $\beta_1 = 0$ eso significa que el valor de x no afecta a Y: este caso lo denotamos con $Y \perp X$, entonces $Y \perp X \Leftrightarrow \beta_1 = 0$.

En general, la variable de respuesta Y puede depender de k variables denotadas por $x_1, \ldots, x_k$:

$$Y = \beta_0 + \beta_1 x_1 + \ldots + \beta_k x_k + \varepsilon. \tag{2.2}$$

en donde $\beta_0, \beta_1, \ldots, \beta_k$ son parámetros desconocidos, ε es una variable aleatoria con esperanza cero y $x_1, \ldots, x_k$ son variables conocidas. La ecuación (2.2) es llamada *modelo de regresión lineal múltiple*, ya que involucra más de una variable predictora. Suponemos que $E(\varepsilon) = 0$ por lo tanto la ecuación (2.2) es equivalente a $E(Y) = \beta_0 + \beta_1 x_1 + \ldots + \beta_k x_k$.

Este modelo es bastante utilizado por su sencilla interpretación: $\beta_i = 0 \Leftrightarrow Y \perp X_i | X_{-i}$, donde denotamos X_{-i} como el vector X dentro del cual se excluye su i-ésima entrada y con $Y \perp X_i | X_{-i}$ referimos que si fijamos X_{-i} entonces X_i no afecta a la respuesta.

Existen muchos modelos que implican términos de orden más alto, algunos con una interpretación fácil. Por ejemplo, considere el siguiente modelo de regresión lineal:

$$E(Y \mid X_1 = x_1, X_2 = x_2) = \alpha + \alpha_1 x_1 + \alpha_2 x_2 + \alpha_{1,2} x_1 x_2, \quad x_i \in \{0, 1\} \tag{2.3}$$

si tomamos $\alpha_{1,2} = -\alpha_1$ entonces

$$E(Y \mid X_1 = x_1, X_2 = x_2) = \alpha + \alpha_1 x_1 (1 - x_2) + \alpha_2 x_2 \tag{2.4}$$

lo cual significa que dado $x_2 = 1$, $E(Y \mid X)$ no depende del valor que tome x_1, eso lo denotamos como $Y \perp X_1 | X_2 = 1$.

El valor de la variable Y depende del valor que tome X, por lo tanto, podemos decir que Y esta condicionada a X. Denotamos en general con $C(Y \mid X)$

la propiedad de interés de la distribución condicional de Y dados los predictores $X = (X_1, \ldots, X_n)$.

El modelo de regresión logístico el modelo de regresión **logit** se fundamenta en una curva en forma de S alargada acotada en el intervalo abierto [0, 1].

Dos importantes elecciones para $C(Y \mid X)$ son $C(Y \mid X) = E(Y \mid X)$ y si Y es una variable binaria $C(Y \mid X) = logit(P(Y = 1 \mid X)) = log(\frac{P(Y=1|X)}{P(Y=0|X)})$.

Definimos un **Modelo CSR** sobre $(Y, X_1, \ldots, X_n)$ como el conjunto de modelos de regresión lineal $C(Y \mid X)$ que satisfacen las restricciones del tipo:

$$Y \perp X_i \mid X_{-i} \tag{2.5}$$

y / ó

$$Y \perp X_i \mid X_j = x_j, X_{-i,-j}. \tag{2.6}$$

La restricción (2.5) se lee: el predictor X_i no afecta $C(Y \mid X)$ para cualquier valor del resto de los predictores: $C(Y \mid X_i = x_i, X_{-i} = x_{-i}) = C(Y \mid X_i = 1 - x_i, X_{-i} = x_{-i})$.

La restricción (2.6) se interpreta como: X_i no afecta $C(Y \mid X)$ a menos que $X_j = x_j$. Más adelante se explica exactamente que representa cada una de las restricciones.

Si X es un vector binario entonces para el caso general tenemos:

$$C(Y|X) = \alpha + \sum_i \alpha_i x_i + \sum_{i \neq j} \alpha_{i,j} x_i x_j + \cdots + \alpha_{1,\ldots,n} x_1, x_2, \cdots, x_n. \tag{2.7}$$

Si hablamos de un **Modelo CSR** general, tomamos todas las combinaciones del conjunto de {1, ..., n} como en (2.7). Si denotamos de nuevo con $x_A = \prod_{i \in A} x_i$, entonces para el modelo general los términos de interacción son:

$$A \subset \{1, \ldots, n\}. \tag{2.8}$$

Ejemplo 1 *Si $n = 3$, entonces para el modelo general tenemos el siguiente conjunto de términos de interación* $\{0, 1, 2, 12, 3, 13, 23, 123\}$.

Un caso particular de la ecuación (2.7) es el modelo llamado de orden dos:

$$C(Y|X = x) = \alpha + \sum_i \alpha_i x_i + \sum_{i \neq j} \alpha_{i,j} x_i x_j. \tag{2.9}$$

Si hablamos de **Modelo CSR** de orden dos, solo tomamos aquellas combinaciones entre a lo más dos números binarios del conjunto de $\{1, \ldots, n\}$. Si denotamos con $x_A = \prod_{i \in A} x_i$, entonces para el modelo de orden dos los términos de interacción son:

$$A \ \subset \{1, \ldots, n\}, \ t.\ q.\ \#A \leq 2. \tag{2.10}$$

Ejemplo 2 *Si $n = 3$, entonces para el modelo de orden dos tenemos el siguiente conjunto de términos de interación* $\{0, 1, 2, 12, 3, 13, 23\}$.

2.2. Gráficas de un Modelo CSR

Como ya se mencionó un **Modelo CSR** se visualiza con base en gráficas, pues a cada modelo se le asocia una gráfica. Por lo cual en esta sección se explican las reglas para su construcción.

Cada X_i con $1 \leq i \leq n$ es asociado con un nodo de la gráfica y Y representa la variable de respuesta. El tipo de conexión entre X_i y Y, simboliza el efecto de X_i en Y (más específico con $C(Y \mid X)$).

Para interpretar las conexiones entre los nodos de la gráfica, tomemos en cuenta la siguiente clasificación:

1. La ausencia de una conexión entre Y y alguna X_i, representa una restricción de la forma (2.5); es decir *no existe dependencia* entre X_i y Y.

2. Una conexión de menor intensidad representa una restricción de la forma (2.6); una arista se forma comenzando desde el predictor X_j hasta la conexión existente entre Y y X_i; llamamos X_j el predictor que controla X_i; ponemos a lado de la arista el valor correspondiente de 1- x_j para indicar la ausencia de una independencia. A esto lo llamaremos una *dependencia parcial* con Y.

3. Una conexión continua representa la ausencia de alguna restricción de la forma (2.5) o (2.6). Lo cual significa una *dependencia total* con Y.

A modo de un pequeño ejemplo, supongamos que se cuenta con un archivo con la siguiente variable de respuesta:

Y El alumno acredito la materia,
y con las siguientes variables de entrada:

X_1 Realizo todas las tareas,
X_2 Puso atención en clases,
X_3 Participo en clases,
X_4 No asistia con regularidad a la clase,
X_5 Presento todos los examenes,
X_6 Presento el proyecto final,
X_7 Obtuvo calificaciones aprobatorias en las evaluaciones parciales.

En base a estas caracteristicas, podemos realizarnos varias preguntas, como por ejemplo:

¿El hecho de que el alumno alla realizado todas sus tareas, dado que no puso atención en clases, influye en el resultado final de que el alumno acredito la materia?. Esta pregunta la podemos representar como:

$Y \perp X_1 | X_2 = 0, X_{-1,-2}$.

¿El hecho de que el alumno pusierá atención en las clases, dado que si realizo todas sus tareas, influye en el resultado final de que el alumno acredito la materia?. Esta pregunta la podemos representar como:

$Y \perp X_2 | X_1 = 1, X_{-2,-1}$.

¿Repercute en la decisión final, si el alumno participaba en clases, dado que faltaba constantemente a ellas?. Esta pregunta la podemos representar como:

$Y \perp X_3 | X_4 = 1, X_{-3,-4}$.

¿Repercute en la decisión final, el hecho de que el alumno presentará todos los examenes, dado que no faltaba contantemente a clases?. Esta pregunta la podemos representar como:
$Y \perp X_5 | X_4 = 0, X_{-5,-4}$.

¿Repercute en la decisión final de si el alumno acredita el curso, si no presento el proyecto final?. Esta pregunta la podemos representar como:

$Y \perp X_6 | X_{-6}$.

Este conjunto de cinco preguntas en particular las podemos representar juntas en una gráfica, tomando en cuenta las clasificaciones establecidas con anterioridad. La Figura 2.1 muestra el grafo correspondiente al conjunto de cinco preguntas.

Para cada gráfica del **Modelo CSR**, el valor de cada conexión toma un color en particular como representación visual del valor. Por ejemplo: la conexión correspondiente a $Y \perp X_1 | X_2 = 0, X_{-1,-2}$ es de color oscuro y para $Y \perp X_2 | X_1 = 1, X_{-2,-1}$ es de color claro. En otras palabras un valor cero lo representamos gráficamente con

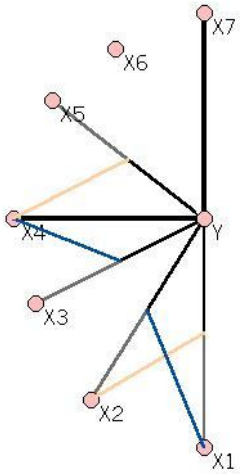


Figura 2.1: Ejemplo de una gráfica del **Modelo CSR**.

un color oscuro y un valor uno se representa con un color claro.

Definición 1 *Una gráfica es regular, si se satisfacen las siguientes dos condiciones [4]*

A1. Una conexión es controlada por no más de un predictor.

A2. Si un predictor X_j controla la conexión entre Y y X_i, debe haber una conexión de menor intensidad o completa entre Y y X_j y el nodo X_i es el único que puede controlar la conexión entre Y y X_j.

La gráfica de un **Modelo CSR** regular, M, la codificaremos por medio de la n-tuple $(\delta_1(M), \ldots, \delta_n(M))$, respetando la clasificación definida al inicio de esta sección, donde

$$\delta_j(M) = \begin{cases} i & \text{si } Y \perp X_j | X_i = 1, X_{-i,-j} \quad (\textit{Dependencia parcial}) \\ -i & \text{si } Y \perp X_j | X_i = 0, X_{-i,-j} \quad (\textit{Dependencia parcial}) \\ * & \text{si } Y \perp X_j | X_{-j} \quad (\textit{Sin dependencia}) \\ 0 & \text{en otro caso} \quad (\textit{Dependencia completa}) \end{cases}$$

Por ejemplo, la gráfica mostrada en la Figura 2.1 es representada por la 7-tuple (-2, 1, 4, 0, -4, *, 0). En esta gráfica podemos observar que X_6 no está conectada

con la variable de respuesta Y, X_7 está conectada con una línea continua y no es predictor de otra variable. Por otro lado, tenemos que X_1 controla a X_2 y viceversa lo cual indica que se trata de una pareja de predictores. Por último vemos que X_4 controla tanto a X_3 como a X_5 y estas a su vez no controlan a ninguna otra variable.

Por la descripción anterior podemos pensar en agrupar los predictores del modelo y así obtener particiones de este, con la finalidad de entenderlos un poco mejor. Con lo siguiente podemos clasificarlo como:

Tipo 1. Conjunto que contiene un predictor, sin conexión con la variable de respuesta.

Tipo 2. Conjunto que contiene un predictor, conectado con la variable de respuesta con una línea continua y que no controla ningún otro predictor.

Tipo 3. Conjunto que contiene una pareja de predictores, que se controlan entre ellos (predictores pareados).

Tipo 4. Conjunto que contiene dos o más predictores donde un único predictor controla al resto del grupo.

Denotamos C_i con $1 \leq i \leq 4$ el conjunto de todos los subconjuntos de tipo i. Retomando el ejemplo de la Figura 2.1 las particiones son: $C_1 = \{\{X_6\}\}$, $C_2 = \{\{X7\}\}$, $C_3 = \{\{X_1, X_2\}\}$, $C_4 = \{\{X_3, X_4, X_5\}\}$.

Con la ecuación (2.9) y respetando las cinco restricciones del ejemplo mostrado en la Figura 2.1, la expresión matemática de este modelo es:

$$C(Y|X) = \alpha+\alpha_7 x_7+\alpha_4 x_4+\alpha_{4,7} x_4 x_7+\alpha_{1,2}(1-x_1)x_2+\alpha_{3,5} x_3 x_5+\alpha_{5,4}(1-x_4)x_5 \quad (2.11)$$

donde el número de α's en el modelo representan los grados de libertad del mismo.

Definimos una gráfica inconsistente, como aquella donde las restricciones que representa se contradicen entre si. Por ejemplo; tomemos en cuenta las siguientes restricciones, representadas en la Figura 2.2.

$$Y \perp X_2 | X_{-2} \Leftrightarrow Y \perp X_2 | X_1 = x_1, \forall x_1 \tag{2.12}$$

$$Y \perp X_1 | X_2 = 0 \tag{2.13}$$

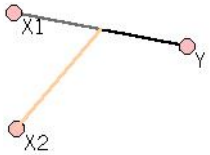


Figura 2.2: Gráfica Inconsistente.

Desglosando las restricciones anteriores tenemos de (2.12) que,

$$\begin{aligned} C(Y|X_2 = 0, X_1 = 0) = &\quad C(Y|X_2 = 1, X_1 = 0) \\ C(Y|X_2 = 0, X_1 = 1) = &\quad C(Y|X_2 = 1, X_1 = 1) \end{aligned}$$

de (2.13) tenemos,

$$C(Y|X_1 = 0, X_2 = 0) = \quad C(Y|X_1 = 1, X_2 = 0)$$

de las restricciones anteriores nos queda,

$$C(Y|X_1 = 0, X_2 = 1) = \quad C(Y|X_1 = 1, X_2 = 1)$$

lo cual nos da la restricción $Y \perp X_1 | X_2 = 1$, lo que contradice a (2.13), por lo tanto la Figura 2.2 que representa las dos restricciones (2.12) y (2.13), es una gráfica inconsistente.

2.3. Caso Particular

Para el caso particular de orden dos de **Modelo CSR**, suponemos inicialmente que $C(Y|X)$ es de la forma (2.9). Por lo tanto,

$$Y \perp X_i | X_{-i} \; sii \; \alpha_i = \alpha_{i,j} = 0, \; \forall \; j \; \neq \; i \tag{2.14}$$

$$Y \perp X_i | X_j = x_j, X_{-i,-j} \; sii \; \alpha_i + x_j \alpha_{i,j} = 0, \alpha_{i,k} = 0, \; \forall \; k \; \neq \; i, j \tag{2.15}$$

Suponemos que todas las interacciones del modelo son de orden dos. Por (2.9) concluimos que

$$Y \perp X_i | X_k = x_k, X_{-i,-k} \ \& \ Y \perp X_i | X_l = x_l, X_{-i,-l} \ \& \ k \neq l \Rightarrow Y \perp X_i | X_{-i} \tag{2.16}$$

$$Y \perp X_i | X_k = x_k, X_{-i,-k} \ \& \ Y \perp X_k | X_l = x_l, X_{-k,-l} \ \& \ i \neq l \Rightarrow Y \perp X_i | X_{-i} \tag{2.17}$$

Propiedad 1 *Una gráfica representa un modelo consistente de orden dos sii las condiciones A1 y A2, de la definición 1, se satisfacen [4].*

Si se desea consultar la prueba para la propiedad 1 puede ser consultada en el apéndice de [4].

Una buena característica de estos modelos es que cada **Modelo CSR** de orden dos es equivalente a un modelo de regresión particular, sin restricciones entre los parámetros. Por lo tanto, un procedimiento de estimación clásica de regresión puede ser usada.

Si $Y \perp X_1 | X_2 = 1$.

$$C(Y|X_2 = x_2, Z = z) = \alpha + \alpha_2 x_2 + \alpha_z z \ con \ z = x_1(1 - x_2) \tag{2.18}$$

Similarmente, si $Y \perp X_1 | X_2 = 0$

$$C(Y|X_2 = x_2, Z = z) = \alpha + \alpha_2 x_2 + \alpha_z z \ con \ z = x_1 x_2 \tag{2.19}$$

En general, cada **Modelo CSR** de orden dos puede ser transformado en un modelo de la forma

$$C(Y|X_A = x_A, Z_B = Z_B) = \alpha + \sum_{i \in A} \alpha_i x_i + \sum_{(i,j) \in B} \alpha_{i,j} z_{i,j} \tag{2.20}$$

donde α_i y $\alpha_{i,j}$ representan los parámetros libres.

2.4. Caso General

Si abreviamos $\alpha_{i,i_1,\ ...,i_k}$ por $x_{i,A}$ donde $A = \{i_1,\ \cdots,\ i_k\}$ de la ecuación (2.7), entonces obtenemos para el modelo general de **Modelo CSR**:

$$Y \perp X_i | X_j = x_j, X_{-i,-j}\ sii\ \alpha_{i,A} + x_j \alpha_{i,j,A} = 0,\ \forall A \subset \{1, \cdots, n\} \backslash \{i, j\}. \qquad (2.21)$$

Contrario al caso particular, un mismo parámetro $\alpha_{i,j,A}$ puede aparecer en varias ecuaciones que simultaneamente crean un sistema de ecuaciones de la forma (2.21). De hecho el sistema debe ser resuelto para obtener una parametrización explícita.

Igual que en la sección anterior, podemos preguntar si todas las gráficas representan un modelo consistente. Para verificar esto considere $(x_1, \ldots, x_n)$ como las esquinas de un hipercubo n-dimensional. La Figura 2.3 muestra un hipercubo de tres dimensiones. Para una independencia dada $Y \perp X_i | X_{-i} = x_{-i}$, denota con $\mathcal{E}(Y \perp X_i | X_{-i} = x_{-i})$ la arista definida por las esquinas $(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$ y $(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$.

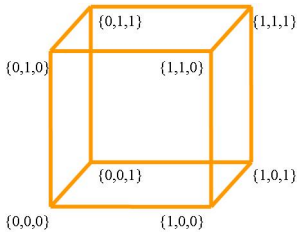


Figura 2.3: Hipercubo 3-dimensional.

Para un conjunto de independencias S, $\{Y \perp X_i | X_j = x_j, X_{-i,-j} = x_{-i,-j}\}$, definimos un camino como una secuencia de aristas $\mathcal{E}(s)$, $s \in S$, tal que aristas subsecuentes siempre tienen una esquina en común. La Figura 2.4 muestra un camino formado por $Y \perp X_1 | X_2 = 1, X_3 = 0$, $Y \perp X_3 | X_1 = 1, X_2 = 1$ y $Y \perp X_2 | X_1 = 1, X_3 = 1$ en un hipercubo 3-dimensional.

Lema 1 *Una nueva independencia es implicada por un conjunto de independencias de S sii la arista correspondiente a la nueva independencia, representa la arista ini-*

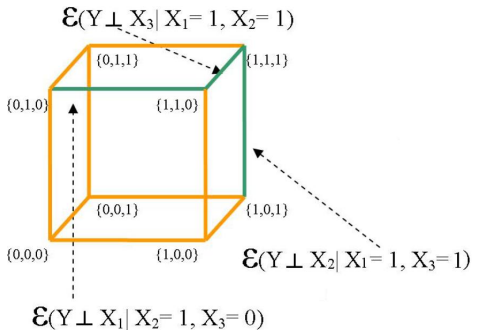


Figura 2.4: Ejemplo de un camino en un Hipercubo 3-dimensional.

cial y final de un camino en S. En otras palabras al agregar una nueva independencia se crea un ciclo [4]. La demostración se puede ver en [4].

Ejemplo 3 *La Figura 2.5 muestra el conjunto de independencias* $S = \{Y \perp X_2 | X_1 = 0, Y \perp X_2 | X_1 = 1, Y \perp X_1 | X_2 = 0\}$*, la restricción* $Y \perp X_1 | X_2 = 1$ *es una independencia implicada de S ya que se forma un ciclo. El ciclo comienza en (0, 1) y finaliza en (1, 1), pasando a través de* $\mathcal{E}(Y \perp X_2 | X_1 = 0)$*,* $\mathcal{E}(Y \perp X_2 | X_1 = 1)$ *y* $\mathcal{E}(Y \perp X_1 | X_2 = 0)$*. Por lo tanto la gráfica correspondiente, Figura 2.5, define un modelo inconsistente.*

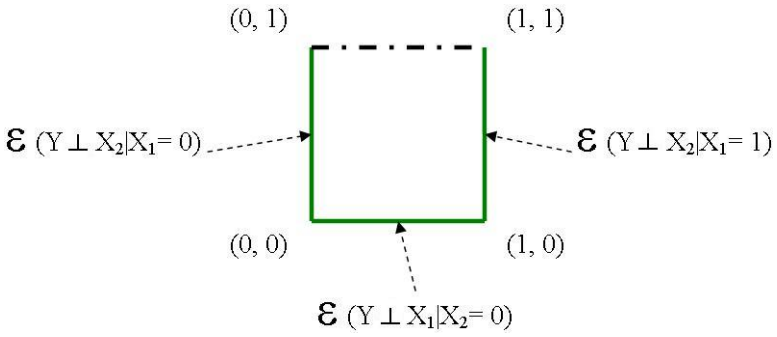


Figura 2.5: Hipercubo 2-dimensional.

Lo anterior forma un algoritmo para verificar si una gráfica es consistente.

Para $n \geq 3$ el hipercubo correspondiente es más complicado, por lo tanto no es fácil detectar a simple vista el hecho de que se forme un ciclo. Por ejemplo, el hipercubo correspondiente a $n = 4$ se muestra en la Figura 2.6, por lo anterior se define la siguiente propiedad.

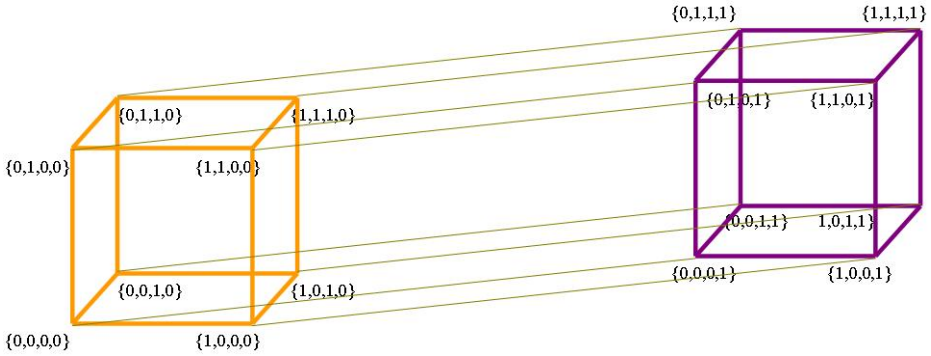


Figura 2.6: Hipercubo 4-dimensional.

Propiedad 2 *Si una gráfica satisface la condición A2 entonces está representa un* ***Modelo CSR*** *general consistente [4].*

2.5. Ajustar un Modelo CSR

Antes de describir el proceso para ajustar un **Modelo CSR** primero necesitamos introducir los conceptos de grados de libertad y de una distribución *JI* cuadrada, así como también el concepto de coeficiente de determinación. Estos conceptos fueron tomados de [7].

Distribución *JI* cuadrada

La distribución *JI* cuadrada es la suma de los cuadrados de variables aleatorias independientes, cada una de las cuales tiene una distribución normal con media cero y varianza uno. Al número de variables aleatorias en la distribución, se le llama los grados de libertad (**gl**).

Suponemos que tenemos una variable aleatoria Z que tiene una distribución normal estándar. Luego la distribución de $U = Z * Z$ es llamada *JI* cuadrada con un **gl**. Generalizado, si $U_1, U_2, \ldots, U_n$ son variables independientes *JI* cuadradas con un **gl**, entonces la distribución $V = U_1 + U_2 + \ldots + U_n$ es conocida como *JI* cuadrada con n **gl** y es denotada por χ^2_n.

La ecuación (2.22) es otra representación matemática de la distribución *JI* cuadrada con $(n-1)$ **gl**, donde n representa el tamaño de una muestra de distribuciones normales, s^2 la varianza muestral y σ^2 la varianza de la población de donde se extrajo la muestra.

$$\chi_n^2 = \frac{(n-1)s^2}{\sigma^2} \tag{2.22}$$

Propiedades de las distribuciones *JI* cuadrada:

- Los valores de *JI* cuadrada son mayores o iguales que 0.
- La forma de una distribución *JI* cuadrada depende de los **gl**. En consecuencia, hay un número infinito de distribuciones *JI* cuadrada.

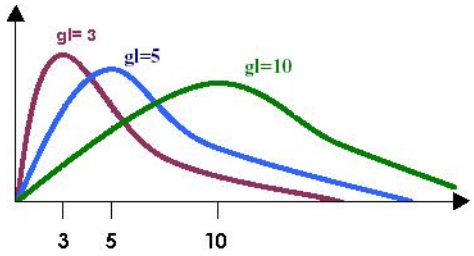


Figura 2.7: Distribuciones *JI* cuadrada con distintos **gl**.

- Las distribuciones *JI* cuadrada no son simétricas. Tienen colas estrechas que se extienden a la derecha; esto es, están sesgadas hacia la derecha.
- La media y varianza de una variable aleatoria *JI* cuadrada con n **gl** son: $E(\chi_n^2) = n$ y $V(\chi_n^2) = 2n$, respectivamente.

Si contamos con un conjunto de distribuciones *JI* cuadrada, seleccionaremos aquella que tenga el valor más pequeño del conjunto de distribuciones.

Coeficiente de determinación

El coeficiente de determinación es una medida de la relación lineal entre dos variables. A medida que su valor es mayor, el ajuste de la recta a los datos es mejor,

puesto que la variación explicada es mayor; así, el desajuste provocado por la sustitución de los valores observados por los predichos es menor, el valor del coeficiente de determinación se representa con R^2. Si todas las observaciones están en la línea de regresión, el valor de R^2 es uno y si no hay relación lineal entre las variables dependiente e independiente, el valor de R^2 es cero.

R^2 es un criterio de valoración de la capacidad de explicación de los modelos de regresión y representa el porcentaje de la varianza justificado por la variable independiente.

$$R^2 = 1 - \frac{Varianza\ no\ explicada}{Varianza\ total} \tag{2.23}$$

Si contamos con un conjunto de coeficientes de determinación seleccionaremos el valor que se aproxime más a uno.

¿Qué significa ajustar un Modelo CSR?

Supongamos que tenemos el conjunto Z (un conjunto es una colección de elementos que se caracterizan en algo común), que está formado por todos los modelos para $P(Y \mid X)$. Dentro de Z tenemos el subconjunto c que está formado por los modelos que cumplan con las independencias de un grafo **Modelo CSR** dado.

Al ajustar un **Modelo CSR**, lo que se busca es encontrar una configuración dentro del subconjunto c que mejor se ajuste a los datos. Para esto se realiza una prueba de hipótesis, donde nuestra hipótesis nula es: "El modelo que mejor se ajusta a los datos", para medir la distancia entre los **Modelo CSR** y la configuración correspondiente al modelo saturado de los datos dentro del conjunto Z, ver Figura 2.8. El **Modelo CSR** que proporcione la menor distancia, será el modelo que mejor se ajuste a los datos.

La distancia la medimos a través de la distribución JI cuadrada de Pearson y tendrá, bajo el supuesto que es el verdadero modelo, una distribución JI cuadrada con n grados de libertad. Por medio de:

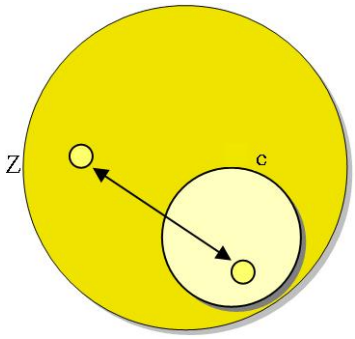


Figura 2.8: Representación gráfica del ajuste de un **Modelo CSR**.

$$\sum \frac{(valor\ esperado - valor\ observado)^2}{valor\ esperado} \sim \chi^2_n. \tag{2.24}$$

Pasos para ajustar un Modelo CSR

Para ajustar un modelo hay que seguir los siguientes pasos:

1. Tomar en cuenta el grafo con todas las conexiones completas.

2. Calcular el impacto de cada uno de los predictores. Para esto se desconecta el i-ésimo predictor $\forall\ i = \{1, \ldots, n\}$ y el resto de los mismos permanecen conectados con Y y se realiza una prueba de hipótesis.

3. Con lo anterior se sabe cual es el predictor de menor importancia. En dado caso de que tengamos algún predictor de poca importancia, procedemos a desconectarlo de la variable de respuesta Y.

4. Probar conexiones entre predictores, simples.

para $1 \leq$ i $\leq$ n

para $1 \leq$ j $\leq$ n

si i $\neq$ j

Evaluar modelo (Y$\perp X_i \mid X_j = 0$)

Evaluar modelo (Y$\perp X_i \mid X_j = 1$)

fin para

fin para

5. Probar conexiones entre predictores, dobles.

para $1 \leq$ i $\leq$ n

para $1 \leq$ j $\leq$ n

si i $\neq$ j

Evaluar modelo (Y$\perp X_i \mid X_j = 0$, Y$\perp X_j \mid X_i = 0$)

Evaluar modelo (Y$\perp X_i \mid X_j = 1$, Y$\perp X_j \mid X_i = 0$)

Evaluar modelo (Y$\perp X_i \mid X_j = 0$, Y$\perp X_j \mid X_i = 1$)

Evaluar modelo (Y$\perp X_i \mid X_j = 1$, Y$\perp X_j \mid X_i = 1$)

fin para

fin para

6. Quitar aquellas conexiones que carezcan de importancia.

2.6. Proceso para obtener el nivel de confiabilidad de un Modelo CSR

En esta sección describiremos un par de ejemplos particulares sobre el proceso que se debe seguir para probar el nivel de confiabilidad de un **Modelo CSR**.

El primer ejemplo que veremos, inicia tomando en cuenta un **Modelo CSR** al azar. Después se aborda el desarrollo necesario para la obtención de la confiabilidad, tanto para el caso particular como para el caso general, utilizando datos binomiales. Para el segundo ejemplo se utilizó datos de tipo lineal.

2.6.1. Primer ejemplo

Veamos un primer ejemplo del uso de **Modelo CSR** para un conjunto de datos específico. Los datos se refiere a un estudio entre 1190 estudiantes de la secundaria

de New Yersey sobre su actitud hacia las matemáticas y el impacto de una serie de conferencias para animar el interés en esa área. El conjunto cuenta con treinta y dos patrones y cinco atributos: el estudiante pone atención a las lecturas, el sexo del estudiante, tipo de escuela, los cursos que prefiere el estudiante y los planes a futuro. Ver Apéndice A.3 para la descripción de los datos, los cuales son de tipo binomial.

Modelo

Tomemos las siguientes restricciones (para n= 5):

$$Y \perp X_1 | X_{-1}$$
$$Y \perp X_2 | X_3 = 0$$
$$Y \perp X_3 | X_2 = 1$$
$$Y \perp X_4 | X_5 = 1$$
$$Y \perp X_5 | X_4 = 1$$

lo anterior representa el modelo (*, -3, 2, 5, 4) y su gráfica correspondiente es:

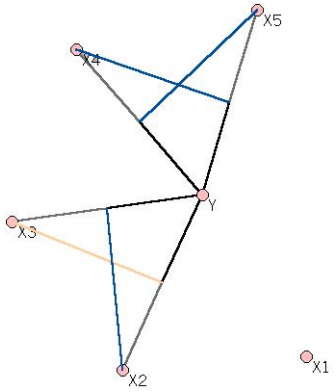


Figura 2.9: Modelo= (*, -3, 2, 5, 4)

Para el caso particular

Tenemos el conjunto de iteraciones posibles {0, 1, 2, 12, 3, 13, 23, 4, 14, 24, 34, 5, 15, 25, 35, 45}. Recordemos que tenemos la restricción $Y \perp X_1 | X_{-1}$, esto significa que las combinaciones donde aparece el número uno no deben de ser contadas, puesto

que X_1 esta desconectada de Y, de lo contrario representaría una contradicción a esa regla, por lo tanto el conjunto de iteraciones posibles para cada restricción nos queda como {0, 2, 3, 23, 4, 24, 34, 5, 25, 35, 45} con este conjunto se aplicará la ecuación (2.21) a cada una de las restricciones del modelo.

Ahora, para la restricción $Y \perp X_2 | X_3 = 0$ nuestro conjunto de iteraciones se reduce un poco más, ya que todas aquellas configuraciones dentro del conjunto que incluyen los valores dos y tres se descartan, con lo anterior tenemos que el conjunto queda como {0, 4, 5, 45}.

Una vez establecido lo anterior se procede aplicar la ecuación $\alpha_{i,A} + x_j * \alpha_{i,j,A} = 0$ (recordemos que se trata de la ecuación (2.21)), donde $i = 2$, $j = 3$, $x_j = 0$ y A representa el conjunto de iteraciones {0, 4, 5, 45}.

Si tomamos $1 \leq k \leq 4$ entonces con $\alpha_{2,A[k]} + 0 * \alpha_{2,3,A[k]} = 0$ obtenemos las cuatro ecuaciones posibles con la restricción $Y \perp X_2 | X_3 = 0$.

Y para la restricción $Y \perp X_4 | X_5 = 1$ nuestro conjunto de iteraciones queda como {0, 2, 3, 23}, para aplicar la ecuación (2.21) tomemos en cuanta que $i = 4$, $j = 5$, $x_j = 1$ y A representa el conjunto de iteraciones {0, 2, 3, 23}. Para $1 \leq k \leq 4$ entonces con $\alpha_{4,A[k]} + 1 * \alpha_{4,5,A[k]} = 0$ obtenemos las cuatro ecuaciones posibles para la restricción dada.

El cuadro 2.1 muestra los resultados obtenidos de la aplicación de (2.21). Cada columna muestra las últimas cuatro restricciones definidas al principio de esta sección (la primera restricción no es aplicable a la ecuación).

Cuadro 2.1: Ecuaciones para el modelo {*, -3, 2, 5, 4}.

$Y \perp X_2 \mid X_3 = 0$	$Y \perp X_3 \mid X_2 = 1$	$Y \perp X_4 \mid X_5 = 1$	$Y \perp X_5 \mid X_4 = 1$
$\alpha_2 = 0$	$\alpha_3 + 1 * \alpha_{2,3} = 0$	$\alpha_4 + 1 * \alpha_{4,5} = 0$	$\alpha_5 + 1 * \alpha_{4,5} = 0$
$\alpha_{2,4} = 0$	$\alpha_{3,4} + 1 * \alpha_{3,2,4} = 0$	$\alpha_{4,2} + 1 * \alpha_{4,5,2} = 0$	$\alpha_{5,2} + 1 * \alpha_{5,4,2} = 0$
$\alpha_{2,45} = 0$	$\alpha_{3,45} + 1 * \alpha_{3,2,45} = 0$	$\alpha_{4,23} + 1 * \alpha_{4,5,23} = 0$	$\alpha_{5,23} + 1 * \alpha_{5,4,23} = 0$
$\alpha_{2,5} = 0$	$\alpha_{3,5} + 1 * \alpha_{3,2,5} = 0$	$\alpha_{4,3} + 1 * \alpha_{4,5,3} = 0$	$\alpha_{5,3} + 1 * \alpha_{5,4,3} = 0$

Despejando las igualdades de la tabla anterior, tenemos:

$$\begin{array}{llll} \alpha_2 = 0 & \alpha_3 = -\alpha_{23} & \alpha_4 = -\alpha_{45} & \alpha_5 = -\alpha_{45} \\ \alpha_{24} = 0 & \alpha_{34} = -\alpha_{234} & \alpha_{24} = -\alpha_{245} & \alpha_{25} = -\alpha_{245} \\ \alpha_{245} = 0 & \alpha_{345} = -\alpha_{2345} & \alpha_{234} = -\alpha_{2345} & \alpha_{235} = -\alpha_{2345} \\ \alpha_{25} = 0 & \alpha_{35} = -\alpha_{235} & \alpha_{34} = -\alpha_{345} & \alpha_{35} = -\alpha_{345} \end{array}$$

de lo anterior obtenemos que:

$$\begin{array}{l} \alpha_4 = \alpha_5 = -\alpha_{45} \longrightarrow \alpha_4 - \alpha_5 = -\alpha_{45} \longrightarrow \alpha_4 - \alpha_5 + \alpha_{45} = 0 \\ \alpha_3 = -\alpha_{23} \longrightarrow \alpha_3 + \alpha_{23} = 0 \end{array}$$

ya que todas las demás igualdades se eliminan entre sí o tiene valor igual a cero, en este caso recordemos que se esta tratando el ejemplo del caso de orden dos, por lo tanto todas aquellas iteraciones que tengan una longitud mayor de dos no son tomados en cuenta.

Ahora bien, como ya se vió, sólo nos quedamos con dos igualdades las cuales restamos a $2^n - 1$ (se le resta uno porque la primera combinación representa un cero), en otras palabras $31 - 2 = 29$ este resultado representa los parámetros libres del modelo completo.

Con estas dos desigualdades hacemos lo siguiente, a $\alpha_4 - \alpha_5 + \alpha_{45} = 0$ la tomamos como $x_4 - x_5 + x_4 * x_5 = 0$ y a $\alpha_3 + \alpha_{23} = 0$ la tomamos como $x_3 + x_2 * x_3 = 0$.

Recordemos que estamos desarrollando este ejemplo con n= 5 y con el conjunto de datos del Apéndice A.3. El conjunto de datos los podemos visualizar como una matriz de treinta y dos renglones por siete columnas. Cada x_i con $1 \leq i \leq n$ representa la i-ésima columna más dos de la matriz de datos.

Esta claro que la operación $x_4 - x_5 + x_4 * x_5$ nos da como resultado un vector de valores al cual renombraremos como $x_1^{'}$ y el vector de valores resultado de la operación $x_3 + x_2 * x_3$ lo renombramos como $x_2^{'}$, tomamos $y_1^{'}$ como el vector de valores tomados de la primera columna de la matriz de datos y a $y_2^{'}$ como el vector de valores tomados de la segunda columna de la matriz de datos.

Con lo anterior definimos $X = (x_1^{'}, x_2^{'})$ y $Y = (y_1^{'}, y_2^{'})$, al calcular el resultado de $C(Y \mid X)$ obtendremos el nivel de confiabilidad del modelo (*, -3, 2, 5, 4).

El último paso es revisar si la gráfica mostrada en la Figura 2.9, correspondiente al modelo en cuestión, es consistente.

Como estamos evaluando para el caso particular basta con aplicar la propiedad 1 para ver si se trata de una gráfica consistente. Dicha propiedad establece que se deben cumplir tanto *A1* como *A2*, de la definición 1, para concluir que se trata de un modelo consistente.

Ambas condiciones se satisfacen en la Figura 2.9, por lo que concluimos que el modelo (*, -3, 2, 5, 4) es consistente para el caso particular.

Para el caso general

Para evaluar este mismo modelo en el caso general, nuestro conjunto de iteraciones posibles, inicialmente, es {0, 1, 2, 12, 3, 13, 23, 123, 4, 14, 24, 124, 34, 134, 234, 1234, 5, 15, 25, 125, 35, 135, 235, 1235, 45, 145, 245, 1245, 345, 1345, 2345, 12345}.

Todo el proceso para calcular el número de parámetros libres y para calcular el nivel de confiabilidad es igual, pero ahora usando el conjunto de iteraciones {0, 2, 3, 23, 4, 24, 34, 234, 5, 25, 35, 235, 45, 245, 345, 2345}, después de eliminar todos los unos involucrados según lo indica la primera restricción. Lo que si cambia es el proceso para verificar si la gráfica es consistente. En este caso se necesita hacer uso de un hipercubo 5-dimensional y buscar si se crean ciclos dentro de él al agregar una nueva independencia.

El cuadro 2.2 muestra algunos de los experimentos realizados, la primera columna representa el modelo expuesto, la segunda indica el tipo de caso (particular ó general), la tercera columna es el valor de los grados de libertad obtenidos para el modelo y por último el valor de la calidad del modelo en cuestión dado por la distribución *JI* cuadrada, representado por χ^2.

Cuadro 2.2: Ejemplo de Modelo CSR, binomial.

Modelo	Caso	gl	χ^2
(*, -3, 2, 5, 4)	particular	29	35.476
(*, 0, *, 5, 4)	particular	29	39.506
(*, -3, 0, 5, 4)	particular	28	32.787
(*, 0, 2, 5, 4)	particular	28	33.467
(*, 5, -5, 5, 0)	particular	27	31.289
(*, -3, 2, 0, 0)	particular	27	34.222
(*, -3, 0, 5, 0)	particular	26	22.575
(*, 0, -5, 5, 0)	particular	26	28.743
(*, -3, 2, 5, 4)	general	28	35.262
(*, 0, *, 5, 4)	general	28	39.184
(*, -3, 0, 5, 4)	general	25	26.433
(*, 0, 2, 5, 4)	general	26	31.620
(*, 5, -5, 5, 0)	general	26	30.907

continuación...			
Modelo	Caso	gl	χ^2
(*, -3, 2, 0, 0)	general	24	25.474
(*, -3, 0, 5, 0)	general	23	20.733
(*, 0, -5, 5, 0)	general	24	23.692

Como ya se indico en la descripción de la distribución de *JI* cuadrada, si tenemos un conjunto de valores correspondientes al resultado dado por dicha distribución, seleccionaremos el menor de los valores listados. Por ejemplo, los resultados expuestos en el cuadro 2.2 para el caso particular seleccionaremos como el mejor modelo que se ajusta a los datos a (*, -3, 0, 5, 0) dado que nos reporta una estimación $\chi^2_{26} = 22.575$ que es el valor más pequeño de entre los correspondientes al caso particular. Para el caso general tomando en cuenta los valores listados en dicho cuadro, el modelo que mejor se ajusta a los datos es (*, -3, 0, 5, 0) con $\chi^2_{23} = 20.733$. Como podemos ver para ambos casos (particular y general) el mejor modelo es el mismo, ya que si el resultado fue bueno con solo iteracciones de orden dos, lo que se espera es que al tomar en cuenta todas las iteracciones posibles el resultado mejore un poco más.

2.6.2. Segundo ejemplo

El segundo ejemplo del uso de **Modelos CSR** que veremos, utiliza datos lineales, los datos corresponden a la ejecución de una red neuronal (este tema se abordará en el Capítulo 5), en el Apéndice A.4 se proporcionan los datos.

Para este ejemplo tomaremos n= 4, lo que significa que para el caso particular el conjunto de iteraciones posibles, inicialmente, es {0, 1, 2, 12, 3, 13, 23, 4, 14, 24, 34} y para el caso general es {0, 1, 2, 12, 3, 13, 23, 123, 4, 14, 24, 124, 23, 134, 234, 1234}.

Todo el proceso es igual al descrito en el primer ejemplo de esta sección, tanto para el caso particular como para el caso general, solo que para datos lineales la calidad del modelo se calcula con R^2, coeficiente de determinación.

El cuadro 2.3 muestra algunos modelos probados y los resultados obtenidos, los modelos han sido seleccionados completamente al azar.

Cuadro 2.3: Ejemplo de Modelo CSR, lineal.

Modelo	Caso	gl	R^2
(*, -3, 2, 0)	particular	77	0.351
(*, 0, 2, 0)	particular	75	0.506
(0, 1, 2, -1)	particular	76	0
(*, -3, 2, 0)	general	76	0.345
(*, 0, 2, 0)	general	74	0.505
(0, 1, 2, -1)	general	75	0.223

Como ya se indico en la descripción de coeficiente de determinación, si tenemos un conjunto de valores correspondientes al resultado dado por R^2, seleccionaremos el valor que más se acerque a uno. Por ejemplo, los resultados expuestos en el cuadro 2.3 para el caso particular seleccionaremos como el mejor modelo que se ajusta a los datos a (*, 0, 2, 0) dado que nos reporta un valor $R^2 = 0.506$ que es el valor más grande de entre los correspondientes al caso particular. Para el caso general tomando en cuenta los valores listados en dicho cuadro, el modelo que mejor se ajusta a los datos es (*, 0, 2, 0) con $R^2 = 0.505$. Como podemos ver para ambos casos (particular y general) el mejor modelo es el mismo, ya que si el resultado fue bueno con solo iteracciones de orden dos, lo que se espera es que al tomar en cuenta todas las iteracciones posibles el resultado mejore un poco más, en este ejemplo en particular el resultado varia muy poco ya que el conjunto de datos tomados para la realización de los experimentos es pequeño.

Capítulo 3

Software para Modelos CSR

En este capítulo se expondrán los métodos utilizados para la implementación computacional del **Modelo CSR**, caso particular y caso general. Adicionalmente se presenta las pantallas proporcionadas por el programa de **Modelo CSR**.

3.1. Caso de Orden Dos

En este apartado describiremos la implementación computacional del caso particular de **Modelo CSR**, comenzaremos con el proceso que se siguio para evaluar la calidad del modelo y sus parámetros libres. Después presentamos el modulo para verificar si el modelo dado representa una gráfica consistente.

Evaluar la calidad del modelo

1. Leer la n-tupla de restricciones correspondientes al modelo.
   Por ejemplo, si tenemos el modelo expuesto en la Figura 3.1, la 5-tuple correspondiente es (2, 0, *, -5, 0).

2. Leer el archivo de datos, proporcionado por el usuario, guardandolo en una matriz.

3. Llamaremos *dataX* a la matriz cuadrada de 2^n, donde n representa el número de variables de entrada del modelo en cuestión. Inicializar *dataX*.

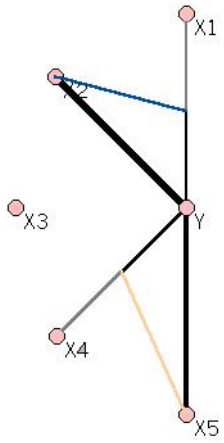


Figura 3.1: 5-tuple= (2, 0, *, -5, 0)

4. Para cada restricción del modelo, modificar los valores de *dataX*.

5. Utilizar la matriz *dataX* y los valores de Y, variable de respuesta, para calcular $C(Y \mid X)$.

Evaluar si se trata de una gráfica consistente

1. Revisar que cada conección de Y con X_i solo esta siendo controlada por a lo más un predictor X_j.

2. Verificar si la conección entre Y y X_j, donde X_j controla la conección entre Y y X_i, esta siendo controlada a su vez por X_i.

3. Si se cumplen los dos pasos anteriores entonces concluimos que la gráfica es consistente.

3.2. Caso General

En este apartado describiremos, en dos partes, la implementación computacional del caso general de **Modelo CSR**, comenzaremos con el proceso que se siguio para evaluar la calidad del modelo y sus parámetros libres. Después presentamos el modulo para verificar si el modelo dado representa una gráfica consistente.

Evaluar la calidad del modelo

1. Leer la n-tupla de restricciones correspondientes al modelo.

2. Generar el conjunto de terminos de iteración, según el conjunto de restricciones, recordemos que si hablamos del caso general, tomamos todas las combinaciones del conjunto de {1, . . . , n}, ver (2.8).

   Ejemplo 4 *Si n= 3 entonces indicamos el conjunto de terminos de iteración como {0, 1, 2, 12, 3, 13, 23, 123}. Para el conjunto de restricciones* $S = \{Y \perp X_1 \mid X_{-1},\ Y \perp X_2 \mid X_3 = 0\}$*, que representan la 3-tuple (*, -3, 0), bajo estas condiciones el conjunto de terminos para el caso general nos queda (0, 2, 3, 23).*

3. Leer el archivo de datos, proporcionado por el usuario, guardándolo en una matriz.

4. Utilizando (2.21) se genera el sistema de ecuaciones correspondiente.

5. El siguiente paso es resolver el sistema. Para resolver el sistema de ecuaciones lineales se implemento el método de *Gauss-Jordan* con intercambio de columnas. Es una variante del método de *Gauss* que es un método de resolución de sistemas de ecuaciones lineales, que consiste en llegar a un sistema "escalonado", transformando la matriz ampliada en una matriz escalonada por filas. Los resultados obtenidos por esté método de resolución de ecuaciones lineales, son utilizados para realizar el calculo final del modelo.

6. Utilizar la matriz resultado del paso anterior, junto con la matriz de datos dada por el usuario, para calcular $C(Y \mid X)$.

Evaluar si se trata de una gráfica consistente

1. Se crea la matriz de adyacencia del hipercubo n-dimensional correspondiente a las restricciones dadas por la n-tuple.

Por ejemplo, para la 3-tuple (*, -3, 0) la matriz de adyacencia correspondiente es:

Cuadro 3.1: Ejemplo de una matriz de adyacencia de un hiercubo 3-dimensional.

	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	0	1	0	0	0	0
3	1	0	0	1	0	0	0	0
4	0	1	1	0	0	0	0	0
5	0	0	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	1
8	0	0	0	0	0	0	1	0

2. Buscar ciclos dentro del hipercubo n-dimensional. Para este paso se siguió el modelo de recorrido de un árbol, utilizando la matriz de adyacencia.

3. Probar si al agregar una nueva linea al hipercubo se genera un ciclo nuevo. Como se puede ver en la Figura 3.2.

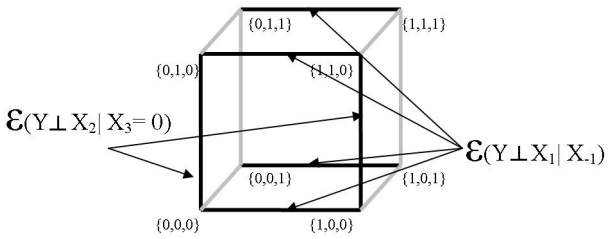


Figura 3.2: Ejemplo hipercubo 3-dimensional.

4. Verificar si el nuevo ciclo formado, corresponde a una restricción del tipo (2.5) o (2.6).

3.3. Pantallas del Software

Esta sección está dedicada a presentar el software desarrollado para la implementación de **Modelo CSR**. El cual como ya se mencionó, fué realizado bajo **R** y utilizando el paquete **Tcl/Tk**.

La primera pantalla es mostrada en la Figura 3.3, donde se le pide al usuario que poprorcione la ruta de un archivo de datos. Así como también indicar si se trata de un archivo binomial o uno de tipo lineal.

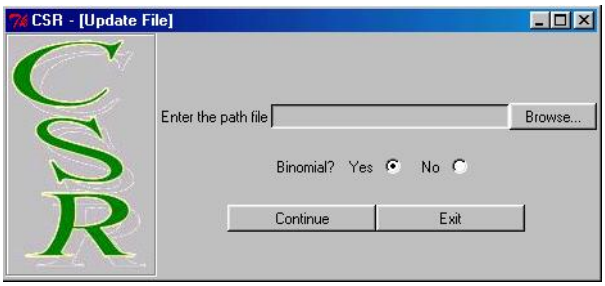


Figura 3.3: Pantalla Inicial.

La segunda pantalla en aparecer es la mostrada en la Figura 3.4, donde podemos crear los modelos a evaluar, arrastrando los predictores, representados por cada X_i nodo, quitando alguna conexion entre X_i con Y (haciendo doble click con el botón izquierdo del mouse sobre el i-ésimo nodo).

La Figura 3.5 muestra la pantalla para agregar nuevas restricciones a un predictor del modelo. El modo de hacer esto, es haciendo click con el segundo botón del mouse sobre el i-ésimo nodo al que se le desea agregar o quitar alguna restricción.

La Figura 3.6 muestra la pantalla generada con un listado de todos los modelos evaluados, desde el inicio de la ejecución del programa. Esté listado se va actualizando (y mostrando) cada vez que se evalua un modelo.

Archivo de datos

El formato del archivo de datos para que el programa pueda leerlo correctamente es el mostrado en la Figura 3.7. Donde del primer renglón, el primer valor

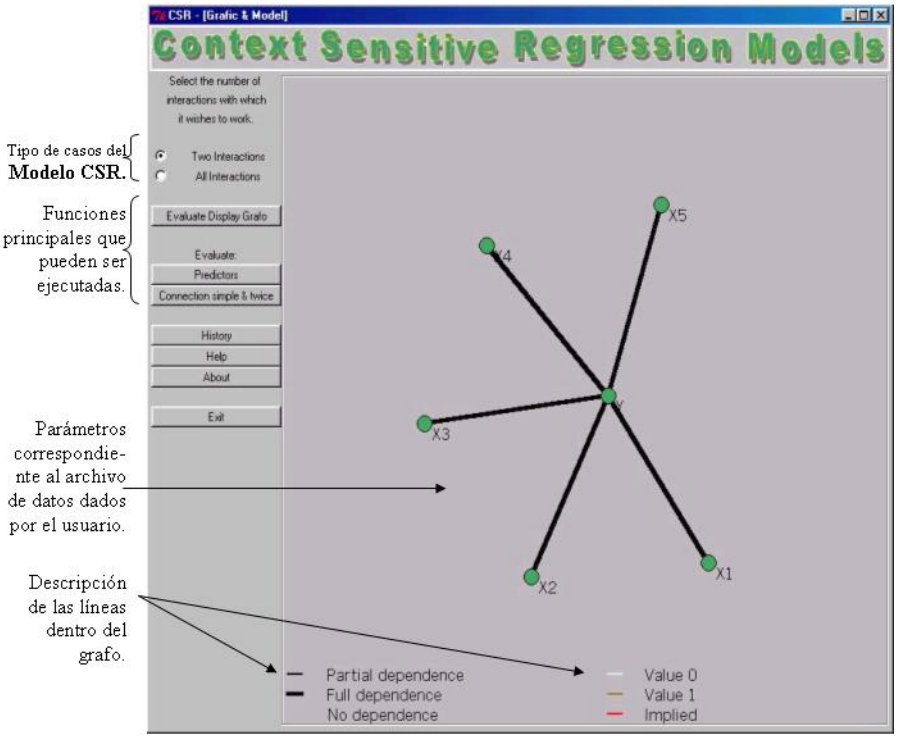


Figura 3.4: Pantalla para crear los modelos.

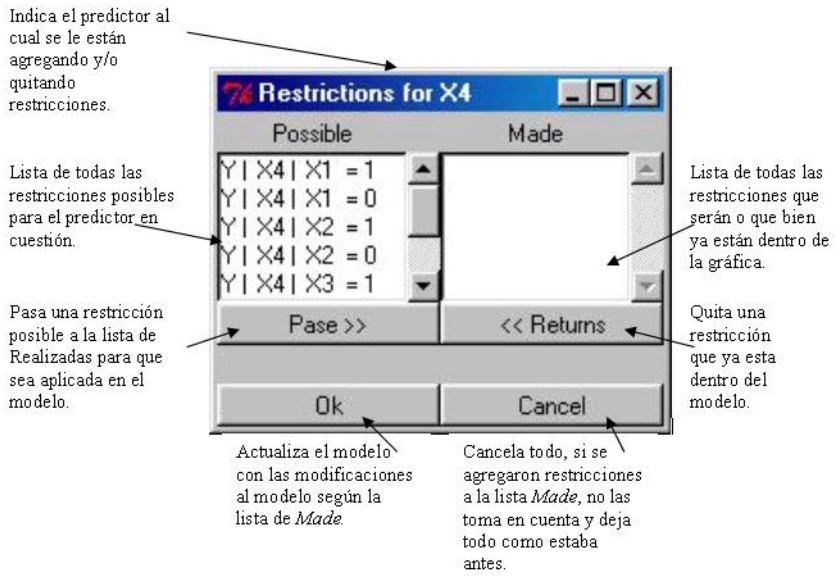


Figura 3.5: Pantalla de Restricciones.

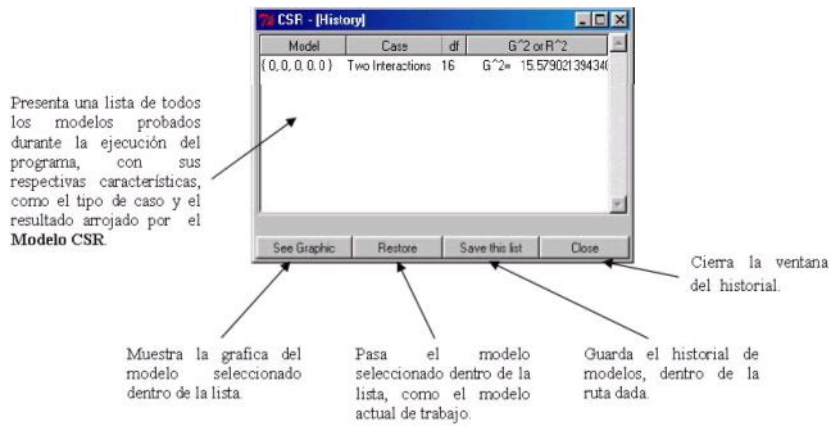


Figura 3.6: Pantalla History.

corresponde al número de columnas (de izquierda a derecha) que corresponden a los valores de respuesta del archivo de datos, y el segundo número (del primer renglón) corresponde al número de variables predictivas con las que cuenta el archivo. Todos los renglones subsecuentes al primer renglón, corresponden a los datos del archivo con los cuales se va a trabajar.

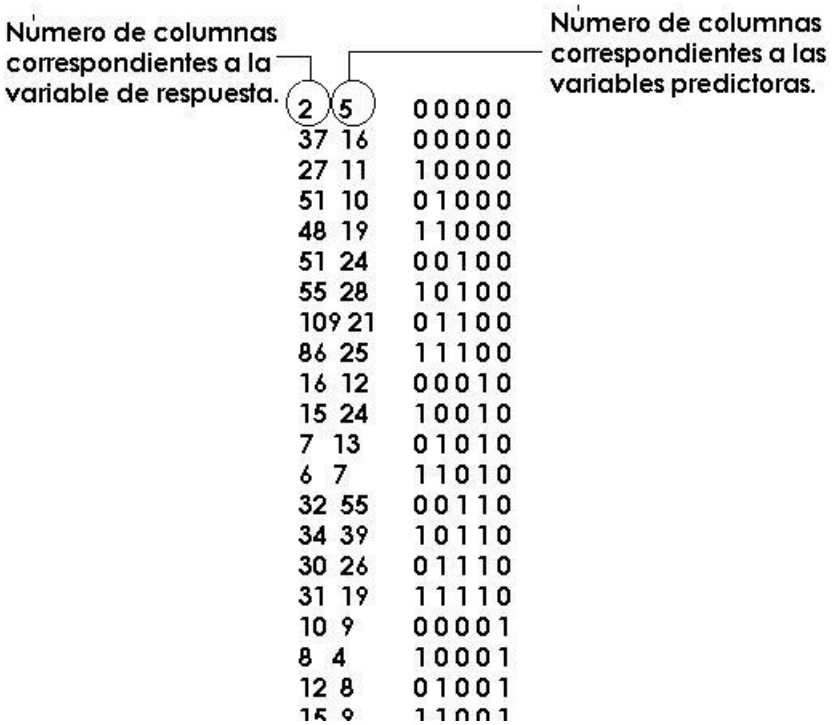


Figura 3.7: Ejemplo del formato de un archivo de datos.

Capítulo 4

Bosques al Azar

El primer método de tipo caja negra que veremos son los *Bosques al Azar* (**BA**), el cual aunque relativamente nuevo, es sin embargo popular por su eficiencia y sencillez.

BA, introducido por Leo Breiman y Adele Cutler en 2001, es una técnica de la *Inteligencia Artificial* para la clasificación de datos, con él se puede construir un clasificador para cualquier tipo de datos. Es un conjunto de árboles de clasificación donde cada árbol trabaja con solo una parte del total de datos destinados para realizar la construcción del bosque. Como en cualquier grupo de trabajo, un bosque depende del buen funcionamiento de cada uno de los árboles así como de la relación entre ellos.

Este método al igual que cualquier otro no es perfecto en el sentido de que, en general, el clasificador generado comete errores de clasificación. Estos errores se pueden cuantificar usando datos y calculando el porcentaje de datos (conjuntos de prueba) mal clasificados, que dependen del poder individual de los árboles y la correlación que exista entre ellos y se disminuyen cuando el números de árboles crece y la correlación entre ellos disminuye, a reserva de que existe un umbral (cantidad de árboles) para el cual la clasificación es la misma a partir de él. En otras palabras, lo más natural es pensar que al aumentar el número de árboles cada vez más, se

obtendran mejores resultados, sin embargo esto no es verdad, ya que al alcanzar cierta cantidad de árboles el resultado de la clasificación no varía.

4.1. Árbol de Clasificación

Se hará parentesis para recordar el funcionamiento de un árbol de clasificación. La meta de los árboles de clasificación es predecir o explicar respuestas en una variable dependiente categórica.

4.1.1. Definición

Un árbol de clasificación es un método de aprendizaje, que tiene una manera muy cómoda y entendible a simple vista de como construir un clasificador a partir de un conjunto de datos, por medio de un diagrama que representa en forma secuencial condiciones y acciones.

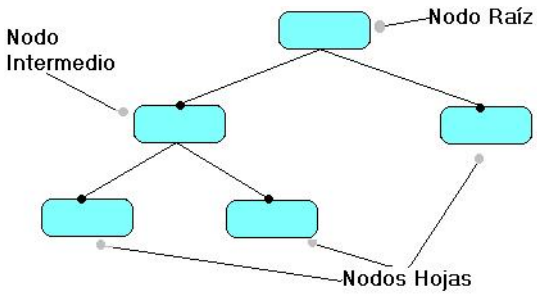


Figura 4.1: Estructura de un árbol de clasificación.

En la Figura 4.1 se puede observar la representación gráfica de un árbol de clasificación. Estos, son construidos a partir de la descripción del problema por resolver. Cada vez que un árbol de decisión se ejecuta, sólo un camino será seguido dependiendo del valor de la variable evaluada.

Los componentes de un árbol de este tipo son: nodos internos o intermedios, nodos hojas y nodo raíz. Cada nodo interno cuenta con una pregunta sobre un atributo

y tiene dos o más hijos, uno por cada posible respuesta, cada nodo hoja tiene una etiqueta de clasificación. El recorrido dentro del árbol comienza por el nodo raíz, el cual realiza la primera pregunta y dependiendo de la respuesta se sigue un camino a lo largo de todo el árbol, hasta alcanzar un nodo hoja.

Un árbol de clasificación parte el espacio H de posibles observaciones en subregiones correspondientes al nivel, entonces cada ejemplo será clasificado por la etiqueta del nodo hoja alcanzado. De esta manera los árboles de decisión pueden ser vistos como un camino jerárquico para describir la partición de H. Un árbol de decisión provee una descripción estructurada de la base de conocimiento. Frecuentemente la misma información puede ser estructurada en otros caminos.

Un conjunto de entrenamiento consiste de datos $(x_1, j_1), \ldots, (x_N, j_N)$ en N casos, donde $x_n \in X$ y $j_n \in \{1, \ldots, J\}$, n= 1, ..., N. El conjunto de entrenamiento es denotado por ℓ

$$\ell = \{(x_1, j_1), \ldots, (x_N, j_N)\} \tag{4.1}$$

Un árbol de clasificación o árbol binario de clasificación, son construidos a partir de cortes iterativos de subconjuntos de X dentro de dos subconjuntos descendientes, iniciando con todos los datos de X.

Los subconjuntos terminales de la partición de X, cada subconjunto terminal es designado por una etiqueta de clase. Dos o más subconjuntos terminales pueden tener la misma etiqueta de clase.

La construcción de un árbol lo podemos resumir como los tres siguientes elementos:

1. La selección de el corte de cada nodo.

2. La desición de el corte de declarar un nodo como terminal o continuar dividiendolo (realizarle otro corte más).

3. El asignarle a cada nodo terminal la etiqueta de la clase a la que corresponde.

La primera parte de la construcción de los árboles de clasificación consiste en la construcción del árbol a partir de un conjunto de prototipos S. Al término de esta primera face, comienza el etiquetado de un conjunto de prueba X, independiente del conjunto de aprendizaje. Se trata de responder las preguntas asociadas a los nodos interiores utilizando los valores de los atributos del patrón X. Este proceso se repite desde el nodo raíz hasta alcanzar una hoja, siguiendo el camino impuesto por el resultado de cada evaluación.

A cada nodo se asocia una pregunta, ¿$X \in A$?, donde A representa el conjunto de los valores que son posibles tomar por X, que cumple:

1. Cada partición depende de un único atríbuto.

2. Si X_i es un atributo categórico, que toma valores en $\{c_1, c_2, \ldots c_n\}$, entonces incluye las preguntas ¿$X_i \in C$?, donde C es el conjunto que alberga a los subconjuntos $\{c_1, c_2, \ldots c_n\}$. Por ejemplo; si tenemos X_2 que toma valores en {Rojo, Verde, Azul}. Las preguntas válidas son del tipo: ¿$X_2 \in \{Rojo\}$?, ¿$X_2 \in \{Verde\}$? y ¿$X_2 \in \{Azul\}$?.

3. Si X_i es un atributo continuo, entonces incluye preguntas del tipo ¿$X_i \leq v$?, donde v es un valor real, v es el punto medio de dos valores consecutivos de X_i. Por ejemplo; si tenemos X_1 toma valores en (0.1, 0.5, 1.0). Las preguntas válidas son: ¿$X_1 \leq (0.1 + 0.5)/2$? y ¿$X_2 \leq (0.5 + 1.0)/2$?.

La principal diferencia entre los algoritmos para la construcción de árboles son las estrategias usadas para podar y la regla para dividir los nodos. Muchos algoritmos solo realizan cortes binarios, que es dividir un nodo en dos y solo unos pocos permiten la posibilidad de varios cortes.

4.1.2. Construcción de un árbol de clasificación

El primer problema en la construcción de un árbol es como usar ℓ para determinar el corte binario de X en piezas pequñas. La idea fundamental para seleccionar el corte de un subconjunto es que los datos en cada subconjunto descendiente sean más puros que los datos de el subconjunto padre. Por ejemplo, si tenemos un conjunto de datos con seis clases, denotamos con $p_1, \ldots, p_6$ la proporción de la clase $1, \ldots, 6$ en cualquier nodo. Para el nodo raíz (denotado por t_1) tenemos $(p_1, \ldots, p_6)$=$(\frac{1}{6}, \frac{1}{6}, \ldots, \frac{1}{6})$. Un buen corte de t_1 será separar las caracteristicas de ℓ tal que todas las características de las clases 1, 2 y 3 vallan al nodo izquierdo y las 4, 5 y 6 vallan al nodo derecho. Ver Figura 4.3.

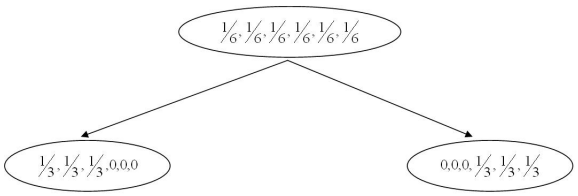


Figura 4.2: Construcción de un árbol de clasificación.

La idea de buscar cortes en los nodos, tal que dan una pureza en nodos descendientes fue implementado como:

1. Definir la proporción de los nodos $p(j/t)$, $j = 1, \ldots, 6$ para ser la proporción de la clase $x_n \in t$ pertenece a la clase j, tal que
   $p(1/t) + \cdots + p(6/t) = 1$.

2. Definir la medida $i(t)$ de la impureza de t como una función no negativa ϕ de $p(1/t), \ldots, p(6/t)$ tal que

   $\phi(\frac{1}{6}, \frac{1}{6}, \ldots, \frac{1}{6}) = maxima$,
   $\phi(1, 0, 0, 0, 0, 0) = 0$,
   $\phi(0, 1, 0, 0, 0, 0) = 0, \ldots$,

$\phi(0,0,0,0,0,1) = 0$.

La impureza del nodo es grande cuando todas las clases estan mezcladas uniformemente y es pequeña cuando los nodos solo contienen una clase.

Para cualquier nodo t, suponemos que δ es un corte candidato de el nodo que lo divide en t_L y t_R tal que la proporción p_L de la clase en t va dentro de t_L y la proporción p_R va dentro de t_R. Ver Figura 4.3.

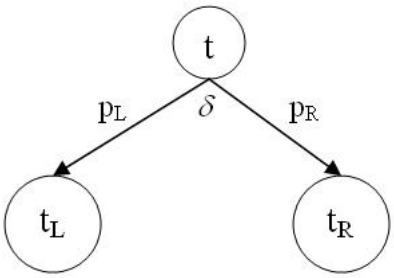


Figura 4.3: Regiones de un árbol de clasificación.

3. Definir un conjunto candidato S de cortes binarios ℓ en cada nodo. Esto es, un conjunto S más simple de cortes como generador inicial, por un conjunto de preguntas κ, donde cada pregunta κ es de la forma:
¿Esta $x \in A$?, $A \subset X$.

El corte δ asociado manda toda x_n en t que responde SI a t_L y toda x_n en t que responde NO a t_R. La impureza del nodo se define como:

$$i(t) = -\sum_{1}^{6} p(j/t)\ log\ p(j/t) \tag{4.2}$$

La construcción es un proceso iterativo empezando con un árbol con una sola hoja. En cada iteración se divide un nodo en dos nuevos nodos (dos regiones nuevas). Con lo anterior nos referimos a particionar un nodo del árbol.

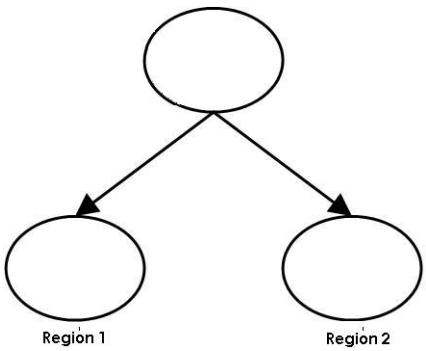


Figura 4.4: Partición de un nodo.

El objetivo de particionar un nodo es incrementar la homogeneidad (en términos de clases) de los subconjuntos resultantes, o lo que es lo mismo, que éstos sean más puros que el conjunto originario. Cada partición tiene asociada una medida de pureza, que se utiliza para la selección de la mejor partición y como criterio de parada.

Resulta más eficiente podar un árbol que detener su crecimiento: la poda permite que un subárbol de un nodo permanezca y la otra desaparezca, mientras que detener el crecimiento poda ambas ramas simultaneamente.

Procedimiento general de poda:

1. Particionar nodos hasta que se cumpla la condición:

   - Que un nodo sea perfectamente homogéneo o que tenga asociados un número de prototipos menor que el umbral dado. Resulta evidente que el resultado debe ser un árbol muy grande, al que se le llamará T_{max}

2. Una vez obtenido T_{max} se trata de podar este árbol, obteniendo una secuencia decreciente y anidada en árboles. Si $T^{'}$ se obtiene a partir de T por poda, $T^{'}$ es un subárbol podado de T y se escribe $T^{'} \prec T$. Así, la secuencia decreciente de árboles podados y anidados será la siguiente

$$T_{max} \succ T_1 \succ T_2 \succ \ldots \succ \{t_1\}$$

de manera que el árbol $\{t_1\}$ es un árbol que consta de un único nodo. Pero solo uno de estos árboles será el que se seleccione finalmente.

Un sencillo criterio para parar y obtener un buen árbol es el siguiente. Un nodo se declarará terminal y en consecuensia no se dividirá, si la clase dominante tiene más del 60 % de los prototipos asociados a ese nodo.

La función de costo involucra un proceso en el que un parámetro de penalización es continuamente incrementado, de manera que la poda se realiza en base a diferentes umbrales. Este proceso da lugar a una secuencia anidada de árboles.

La función de costo se define como:

- Para cualquier subárbol T se define su complejidad como el número de nodos terminales, $\mid \widehat{T} \mid$.
- El error de clasificación asociado al árbol T es $R(T)$.
- La función de costo asociada al árbol T, $R\alpha(T)$, se calcula:

$$R\alpha(T) = R(T) + \alpha \mid \widehat{T} \mid \tag{4.3}$$

donde α es un valor real ($\alpha \geq 0$) (parámetro de complejidad) que se interpreta como el costo de complejidad por nodo terminal.

Así, $R\alpha(T)$ es una combinación lineal del costo del árbol y su complejidad, ponderada apropiadamente. Un valor alto de α produce un incremento en el costo por nodo terminal, lo que se traduce en penalizar un subárbol con un alto número de nodos terminales.

Para realizar una selección entre los árboles se asocia una medida de error a cada árbol de la secuencia y se escoje aquel que tenga asociado el menor error. La idea básica subyacente es la de podar aquellos árboles que produzcan pequeños beneficios de bondad. Se espera que árboles podados produzcan mejores resultados que los

obtenidos con árboles más grandes al clasificar patrones independientes, esto es, los árboles podados tendrán más capacidad de generalización al no estar tan ajustados al conjunto de aprendizaje.

La Figura 4.5 muestra un ejemplo del funcionamiento de un árbol de clasificación, usando los famosos datos Fisher, este conjunto de datos describen tres especies diferentes de flores, en el Apéndice A.2 se encuentra una descripción más amplia.

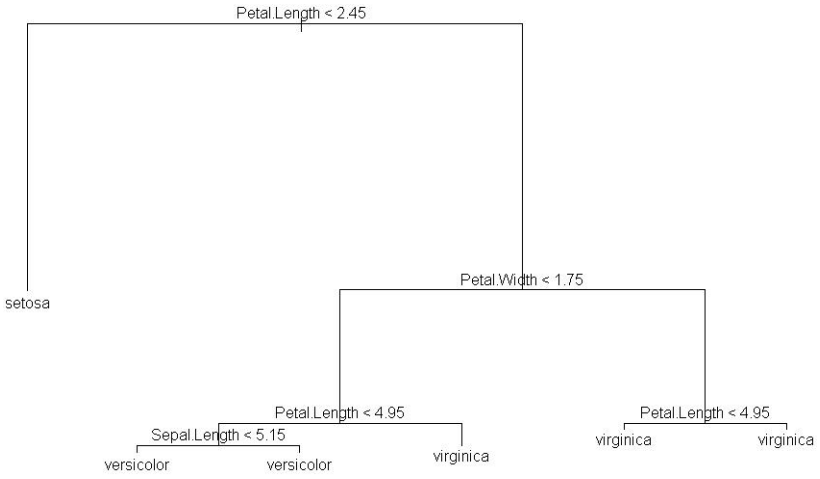


Figura 4.5: Árbol de Clasificación.

4.2. Descripción de un Bosque

BA se crea en base a un conjunto de árboles de clasificación, donde cada nodo del árbol tendrá máximo dos ramificaciones.

Cada árbol del **BA** se construye de la siguiente manera

- Se toma una muestra aleatoria de tamaño N del conjunto original de datos destinados para el entrenamiento del **BA**.
- Si m representa el número total de variables de entrada X, $X = \{X_1, \ldots, X_m\}$ entonces se realiza una selección aleatoria con reemplazamiento para tomar un

subconjunto $S_M(X)$ donde M es el número, menor que m, de variables de entrada con las cuales va a trabajar el árbol. $S_M(X)$ corresponde a tomar ciertas columnas de la matriz de datos, destinados para el entrenamiento.

- Cada árbol puede crecer lo más que pueda, por eso no existen podas.

Para construir un bosque hay que especificar algunos parámetros los cuales a su vez sirven para crear cada uno de los árboles.

- Especificar el número de árboles con los que contará el bosque, por omisión **BA** crea 500 árboles para formar el bosque. Este valor no es ningún delimitador, así que se puede crear un bosque con más o menos árboles.
- Proporcionar un archivo para el entrenamiento del **BA**.
- Proporcionar un archivo de prueba.
- Especificar el valor de m, que representa el número de variables de entrada.
- Especificar el valor de M que representa el número, menor que m, de variables de entrada con las cuales va a trabajar el árbol, por omisión **BA** tiene el valor M= $\sqrt{m}$.

Para clasificar un nuevo objeto $\mathbf{x}$, se pasa $\mathbf{x}$ a cada uno de los k árboles, que conforman el **BA**, estos dan su único voto para la clasificación del objeto $\mathbf{x}$. Los votos se van almacenando y se selecciona aquella clasificación que tuvo el mayor número de votos para $\mathbf{x}$, según el total de árboles dentro del bosque.

Por voto debemos de entender que cada árbol indica la clasificación a la cual pertenece el patrón que se esta evaluando. Por ejemplo, si se tienen cinco árboles, tres patrones y dos clasificaciones posibles para cada patrón de entrada. Para el primer patrón de entrada cada árbol reporta una clasificación para él, así como para el segundo patrón y para el tercer patrón. La Figura 4.6 representa lo anterior, donde podemos ver que para el primer patrón se tienen tres árboles que indican que pertenece a la clase uno y el resto indica que pertenece a la clase dos, entonces se

toma la clase que más se repite, en este caso, la clasificación para el primer patrón es que pertenece a la clase uno, la clasificación para el segundo patrón es que pertenece a la clase dos y la clasificación para el tercer patrón es que pertenece a la clase uno.

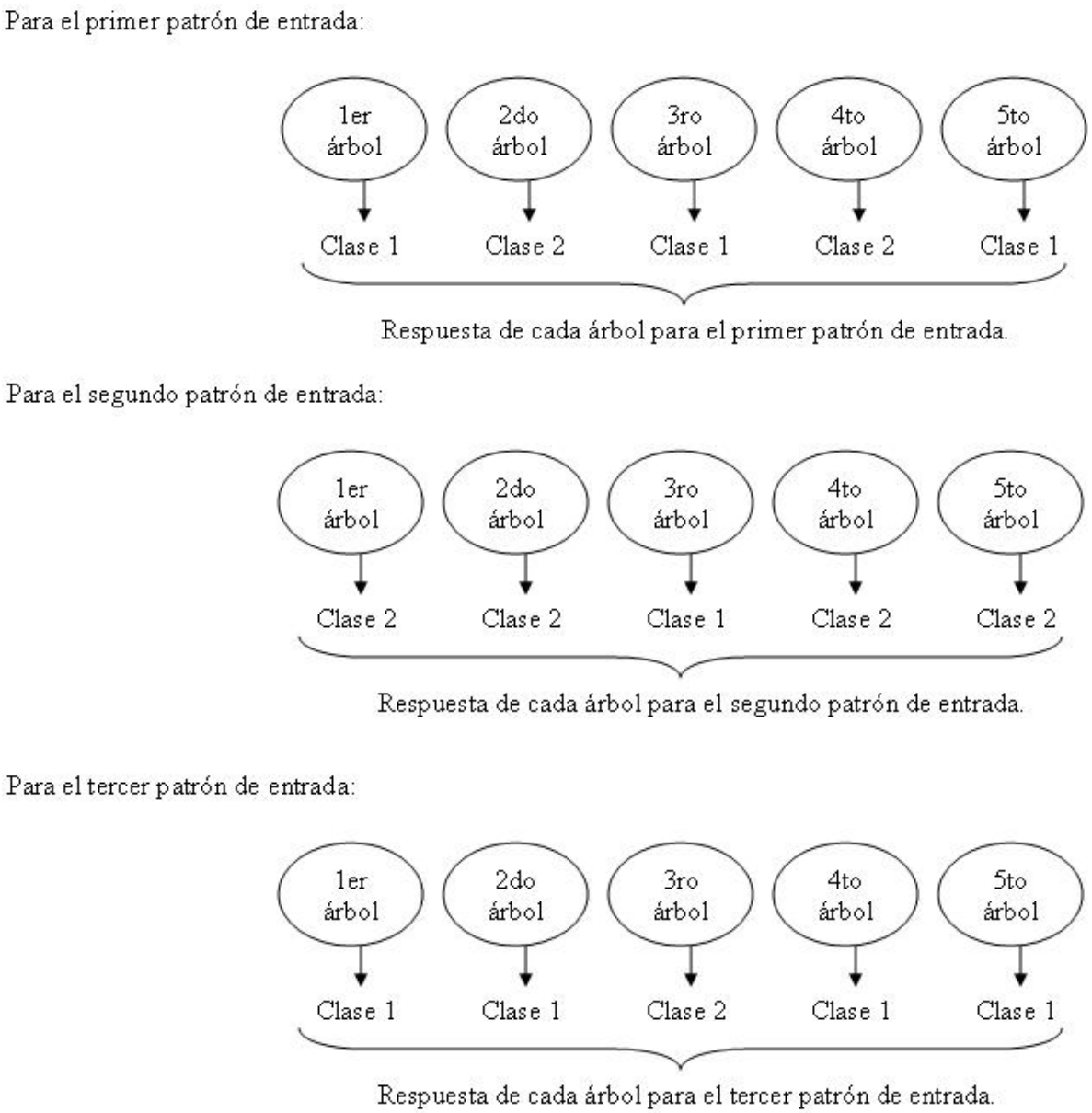


Figura 4.6: Votos del bosque.

4.3. Calidad

La calidad es el porcentaje de error para la clasificación de cierto conjunto de datos por el bosque.

Los datos destinados para el entrenamiento del bosque se divide en n subconjunto, donde n representa el número de árboles del bosque, a cada subconjunto le llamaremos bolsa de datos para el i-ésimo árbol $1 \leq i \leq n$.

De cada bolsa se toman $\frac{2}{3}$ de datos y con estos se construye el i-ésimo árbol. El $\frac{1}{3}$ de datos que fueron descartados de la bolsa, se depositan en un contenedor denominado *Out-Of-Bag* (**OOB**), que literalmente significa fuera-de-la-bolsa. Dentro de este receptáculo de datos, se van agregando todos aquellos datos eliminados de la bolsa de construcción para cada i-ésimo árbol.

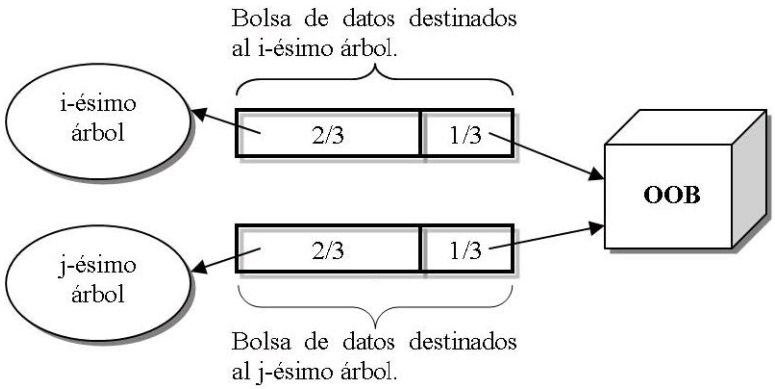


Figura 4.7: Bolsa de datos para cada árbol.

Una vez realizado el proceso de aprendizaje de los árboles, con sus correspondientes $\frac{2}{3}$ de datos, se procede a probar su eficiencia. Para esto se utilizan los datos contenidos en **OOB**, los cuales se pasan uno a uno a los árboles, a excepto del $\frac{1}{3}$ de datos agregados a **OOB** por el árbol en cuestión.

Por esta razón, los **BA** no necesitan de validación cruzada ya que se puede decir

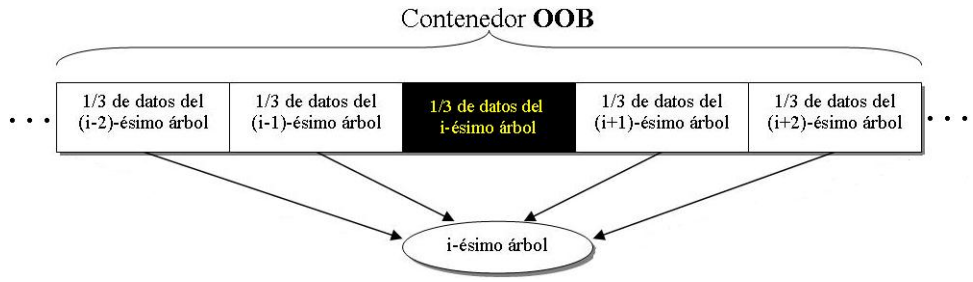


Figura 4.8: Método para probar la eficiencia del i-ésimo árbol.

que este procedimiento se realiza internamente. Por lo tanto **OOB**, calcula la generalización del error, en otras palabras, calcula el error de la clasificación al momento de entrenar.

El porcentaje del error arrojado por el bosque depende de dos factores importantes de cada árbol

1. *La Correlación* que existe entre dos árboles dentro del bosque. El aumento de la Correlación aumenta el porcentaje de error del bosque.

2. *El Poder* de cada árbol dentro del bosque. El *Poder* que tiene cada árbol dentro del bosque está en función del porcentaje de error que tenga para clasificar; si es un mal clasificador su poder es pequeño, en caso contrario es un buen clasificador y cuenta con un poder alto dentro del bosque. Si el *Poder* de los árboles disminuye, entonces el porcentaje de error de todo el bosque aumenta.

Correlación y Poder

El número de variables seleccionadas aleatoriamente para el corte de cada nodo de los árboles, es el único parámetro que **BA** solicita que se especifique, aunque tiene un valor por omisión, $\sqrt{m}$. No es recomendable utilizar este valor de M, más bien se recomienda jugar con diferentes valores hasta encontrar el que nos de un error **OOB** menor. En la literatura se menciona que si disminuimos el valor de M obtendremos una disminución tanto del error de *Correlación* como del *Poder*, en caso contrario tendremos un aumento en ambos, lo cual se reflejaría en el error de

OOB y el error del conjunto de prueba. Sin embargo, hemos jugado con diferentes valores y no vimos que esto realmente se cumpla.

Veamos esto utilizando el conjunto de datos descritos dentro del Apéndice A.1. El cuadro 4.1 muestra la distribución de los datos, los cuales fueron formados de manera aleatoria. En la Figura 4.9 se puede apreciar el comportamiento del entrenamiento del *Bosque al Azar*, que se mide por el error de **OOB** y la predicción del conjunto de prueba.

Cuadro 4.1: Distribución de datos; Indios Pimas.

	Número de datos
Entrenamiento	614 = 80 %
Prueba	154 = 20 %
Total de datos	768 = 100 %

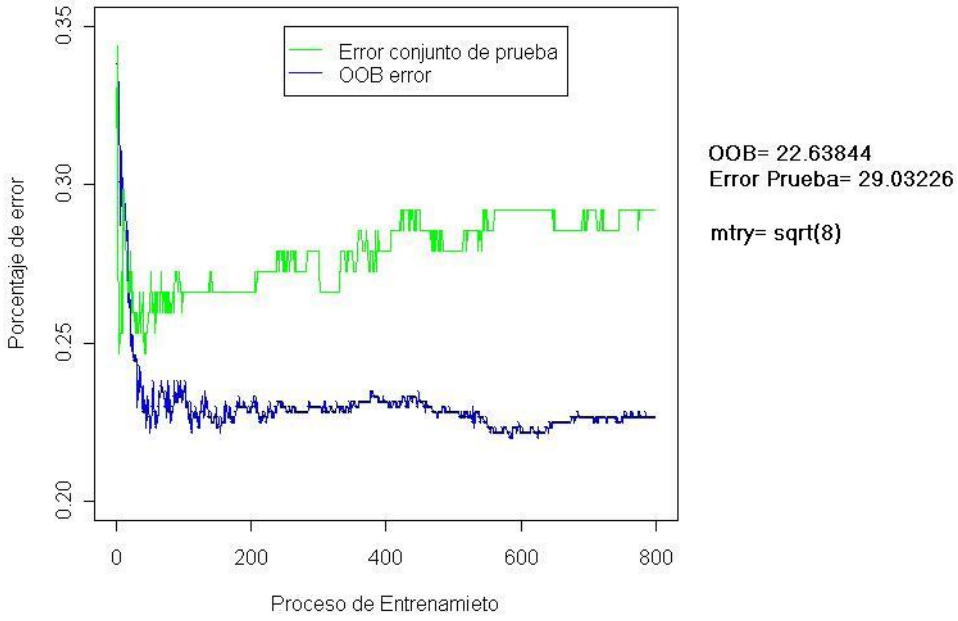


Figura 4.9: Conjunto de Prueba y **OOB**, con $M = \sqrt{8}$.

En la Figura 4.9 se utilizó el valor por omisión de M y se obtuvo un error de estimación para **OOB** de 22.638 % y el error del conjunto de prueba de 29.032 %. La gráfica representa en el eje vertical el porcentaje de error y en el eje horizontal muestra el proceso de entrenamiento de **BA** y la predicción del mismo sobre un conjunto nuevo. En la gráfica se muestra lo que ocurre conforme se van agregando árboles al bosque tanto en el entrenamiento como en la predicción. Como se puede ver, el error de **OOB** oscila en un rango específico cada vez que se agregan mas árboles, en cambio el error del conjunto de prueba disminuye primero y aumenta después. El eje horizontal no es la representación de árboles individuales, sino el comportamiento del bosque según va creciendo, en otras palabras el punto 800 no significa el árbol número 800 y su predicción, sino como ha evolucionado el entrenamiento y la predicción cuando el bosque tiene 800 árboles.

Cuadro 4.2: Pruebas con M.

M	OOB	Prueba
$\sqrt{8}$	22.63844 %	29.03226 %
$\sqrt{8}$ + 1	21.98697 %	28.38710 %
$\sqrt{8}$ - 1	21.66124 %	27.74194 %
$\sqrt{8}$ + 2	22.31270 %	27.74194 %
$\sqrt{8}$ - 2	23.12704 %	27.74194 %

En el cuadro 4.2, se muestran varias pruebas que se realizaron a **BA** modificando el valor de M, para dichas pruebas, se siguió utilizando los mismos conjuntos tanto para el entrenamiento como para la predicción, así como también el mismo número de árboles para el bosque. Puesto que nuestro conjunto de datos es pequeño no vemos cambios muy significativos al jugar con el valor de M, aún así es interesante ver las mejoras que se pueden obtener con la manipulación de un solo componente.

4.4. Variables Importantes

Si tenemos m variables de entrada para nuestra clasificación ¿Realmente se necesitan todas?. Es interesante saber qué pasa con las variables, cuáles de ellas son las que realmente se necesitan y cómo afectan en la predicción de este modelo de tipo caja negra.

Para calcular cuales variables son importantes **BA** tiene su propio método, el cual funciona por medio de permutaciones.

Para estimar la importancia de la i-ésima variable, se permutan los valores de ésta de forma aleatoria y cada árbol le asigna una clasificación, el resultado es "guardado". Al final de la ejecución, el número de votos para la categoria, según la permutación de la i-ésima variable, es comparada contra su valor antes de realizar dicha permutación y la diferencia se considera como la importancia de la i-ésima variable.

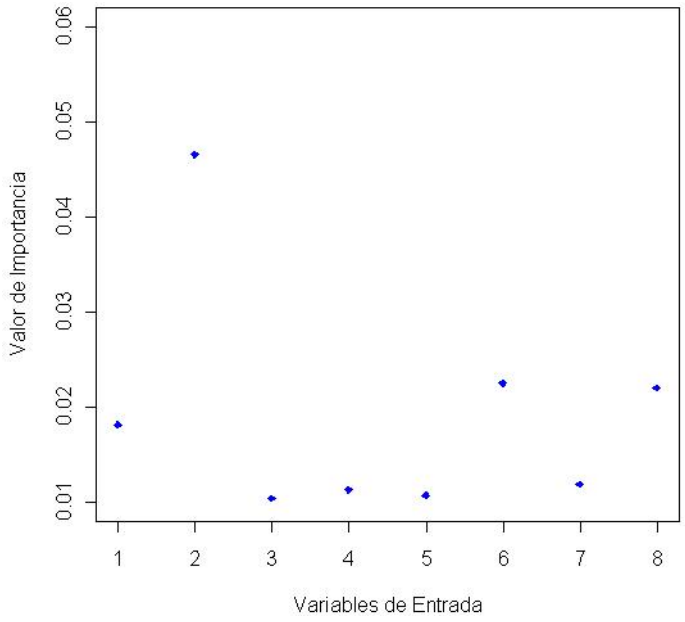


Figura 4.10: Variables Importantes

Continuando con el mismo conjunto de datos, la Figura 4.10 muestra las vari-

ables de entrada contra la importancia de cada una de ellas. Esta gráfica se obtuvo con 800 árboles, M= $\sqrt{8}$, sin un conjunto de prueba. Esta gráfica da como resultado que las variables importantes son: 2, 6 y 8, con un porcentaje del error de estimación **OOB** de 23.31 %.

Ahora veamos como se comporta **BA** usando solo las variables de entrada establecidas como importantes. Usando solo la variable número 2 para la clasificación, el porcentaje del error de estimación **OOB**= 29.43 %, incluyendo la variable número 8 (usando 2 y 8), el porcentaje del error de estimación **OOB**= 28.78 % y finalmente agregando la variable número 6 (usando 2, 6 y 8), el porcentaje del error de estimación **OOB**= 26.04 %. Como se puede observar, utilizando sólo las tres variables catalogadas como importantes se obtienen buenos resultados para la clasificación.

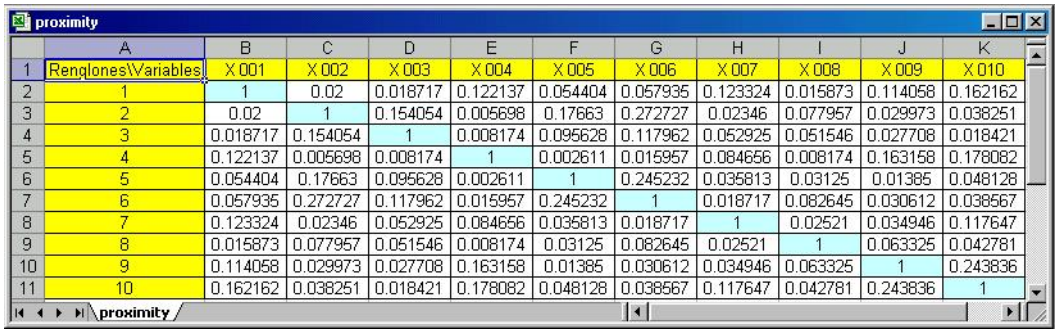
BA solo utiliza permutaciones de una y dos variables de entrada, lo que podemos ver como un punto menos a la eficiencia de este método ya que es incapaz de poder permutar todas las entradas al mismo tiempo y comprobar con esto la importancia de las variables.

4.5. Proximidades

BA implementa un método para definir proximidad entre dos observaciones. El método se describe a continuación.

- Inicializa las proximidades a cero.
- Para cualquier árbol, aplicar el árbol a todos los casos.
- Si en el i-ésimo árbol, los casos k y n estan en el mismo nodo terminal, aumenta la proximidad entre k y n a uno.
- Acumular todo sobre los árboles del bosque y normalizar estas proximidades dividiéndo entre el número de árboles.

El resultado es una matriz cuadrada de dimensiones igual al número de patrones de ejemplos en el archivo de entrenamiento, la cual proporciona la medida intrínseca de proximidad. Donde la medida se define claramente para cualquier tipo de variables independientes, incluyendo las categóricas.



proximity

	A	B	C	D	E	F	G	H	I	J	K
1	Renglones\Variables	X 001	X 002	X 003	X 004	X 005	X 006	X 007	X 008	X 009	X 010
2	1	1	0.02	0.018717	0.122137	0.054404	0.057935	0.123324	0.015873	0.114058	0.162162
3	2	0.02	1	0.154054	0.005698	0.17663	0.272727	0.02346	0.077957	0.029973	0.038251
4	3	0.018717	0.154054	1	0.008174	0.095628	0.117962	0.052925	0.051546	0.027708	0.018421
5	4	0.122137	0.005698	0.008174	1	0.002611	0.015957	0.084656	0.008174	0.163158	0.178082
6	5	0.054404	0.17663	0.095628	0.002611	1	0.245232	0.035813	0.03125	0.01385	0.048128
7	6	0.057935	0.272727	0.117962	0.015957	0.245232	1	0.018717	0.082645	0.030612	0.038567
8	7	0.123324	0.02346	0.052925	0.084656	0.035813	0.018717	1	0.02521	0.034946	0.117647
9	8	0.015873	0.077957	0.051546	0.008174	0.03125	0.082645	0.02521	1	0.063325	0.042781
10	9	0.114058	0.029973	0.027708	0.163158	0.01385	0.030612	0.034946	0.063325	1	0.243836
11	10	0.162162	0.038251	0.018421	0.178082	0.048128	0.038567	0.117647	0.042781	0.243836	1

proximity

Figura 4.11: Matriz de proximidad.

La descripción del conjunto de datos utilizados para ejemplificar este apartado se puede consultar en el Apéndice A.6, dicho conjunto cuenta con 208 patrones de datos, por lo tanto, la matriz de proximidades resultado de la ejecución de **BA** tiene las dimensiones de 208 renglones por 208 columnas.

Esta matriz proporciona información sobre los datos y se utiliza para recuperar valores faltantes y presenta información sobre valores atípicos. Las observaciones que son semejantes tendrán proximidades a uno y los casos más disímiles están más cerca a cero. La Figura 4.11 muestra la sub-matriz cuadrada de los primeros diez registros de la matriz de proximidades, donde podemos ver la proximidad que existe entre los primeros diez patrones de entrada, por ejemplo el primer patrón tiene proximidad uno hacia si mismo, mientras que del resto de los patrones tiene mayor proximidad con el décimo registro a 0.162162 y se encuentra más disímil del octavo patrón con 0.015873.

La matriz de proximidades se calcula al principio de la ejecución del bosque. Para después ser utilizada en caso de ser requerida.

4.6. Datos Faltantes

BA cuenta con dos mecanismos diferentes para reemplazar datos faltantes en el conjunto de entrenamiento y dos más para el reemplazamiento dentro del conjunto de prueba.

- Datos faltantes en el conjunto de entrenamiento

    1. Si la i-ésima variable no es categórica, se calcula la media de los valores de esta variable para la clase j, este valor es usado para reemplazar los datos faltantes de la i-ésima variable dentro de la clase j. En caso contrario (que la i-ésima variable sea categórica), el reemplazo es por la moda de la i-ésima variable para la clase j. A estos valores reemplazados se les da el nombre de **fills**.
    2. Si existe una gran cantidad de valores faltantes, entonces se comienza haciendo un remplazamiento al azar de los valores faltantes. Después de esto se ejecuta el método de proximidades [ver Sección 4.5].

- Datos faltantes en el conjunto de prueba

    1. Si se calcularon los valores reemplazados llamados **fills** (estos valores se obtienen si existen datos faltantes en el conjunto de entrenamiento, ver el punto anterior) entonces se utilizan para reemplazar los datos faltantes, según la etiqueta de la clase.
    2. Si no existen los valores reemplazados llamados **fills**, entonces cada caso en el conjunto de prueba es duplicado tantas veces como clases existan en el conjunto de entrenamiento. La primer copia de un caso es asumido con el valor de la primer etiqueta, lo que significa que los valores faltantes para la primer copia son reemplazados con el valor de la primer etiqueta y así sucesivamente para las demás copias.

Para la realización de este documento se utilizó el *Lenguaje de Programación Estadístico R*, **BA** es un paquete más para este lenguaje. La versión de **BA** con la

que se trabajó es 4.5-1, para la cual aún no se implementan los mecanismos para reemplazar valores faltantes. Es por esta razón, que al intentar verificar esta parte, no se obtuvieron los resultados esperados según la literatura, dentro del manual de usuario de este paquete, bajo *R*, se menciona que en una futura versión se incluirá.

4.7. Paso a paso

En esta sección describiremos los pasos a seguir para crear un **BA** y como funciona bajo R.

Teoría de Breiman

- Se establece el archivo de entrenamiento (y un archivo de prueba, si se desea).
- Se indica el número de árboles para el bosque.
- Se indica el número de variables aleatorias seleccionadas para cada nodo.
- Selecciona las dimenciones de la matriz de datos con los cuales se creará el árbol (esto para cada árbol).
- Se crea el **OOB** según el $\frac{1}{3}$ de datos que sobraron de la matriz de datos destinada para el árbol (para cada árbol).
- Se utilizan los datos contenidos en **OOB**, para probar la eficiencia de cada árbol.
- Si existe un archivo de prueba, se pasa al bosque para que establezca una clasificación para los datos.

Utilizando R

- Se proporciona el archivo de entrenamiento (y un archivo de prueba, si se desea).
- Se establece el valor de M o bien, se deja el valor por omisión.

- Se indica el número de árboles que conformarán el bosque o bien, se deja el valor por omisión.

Nota: La función bajo **R** para realizar el entrenamiento del bosque es la siguiente.

```
randomForest(x, y= NULL, xtest= NULL, ytest= NULL, ntree= 500,
 mtry= sqrt(ncol(x)), importance= FALSE, do.trace= FALSE, ...)
```

donde,

x: Representa la matriz de datos para entrenamiento o bien la formula que describe el modelo que sera utilizado para realizar el entrenamiento del bosque.

y: Representa el vector de respuestas de x.

xtest: Representa la matriz de datos del conjunto de prueba.

ytest: Representa la respuesta del conjunto de datos $xtest$.

ntree: Representa el número de árboles con los cuales se va a construir el bosque.

mtry: Número de variables aleatorias, candidatas para cada corte.

importance: ¿Debe la importancia de predictores ser determinada?.

do.trace: Da más información hacerca de la construcción de los árboles.

...: Indica la posibilidad de agregar más opciones.

Capítulo 5

Redes Neuronales Artificiales

El segundo método de tipo caja negra que trataremos serán las *Redes Neuronales Artificiales* (**RNA**), las cuales, han sido muy utilizadas en computación dentro del área de la *Inteligencia Artificial*. Podemos encontrar varias referencias sobre **RNA**, por esta razón nos limitaremos a describir la arquitectura de red *unidireccional* [5, 10] implementando el modelo *Perceptrón Multi-Capa* (**PMC**), que es el modelo de red más utilizado para el tipo de arquitectura mencionada.

5.1. Descripción

Las **RNA**, originalmente, estaban inspiradas en el comportamiento biológico del cerebro, tratando de imitar su comportamiento. La Figura 5.2 muestra la representación gráfica de una neurona biológica.

Las neuronas y las conexiones que existen entre ellas (conexiones sinápticas) constituyen la clave para el procesador de la información. Las neuronas poseen una estructura de árbol denominada dendritas, reciben las señales de entrada que vienen de otras neuronas a través de las uniones llamadas sinápsis. Las partes en una neurona biológica son:

- Cuerpo de la neurona. Donde se procesa la información.

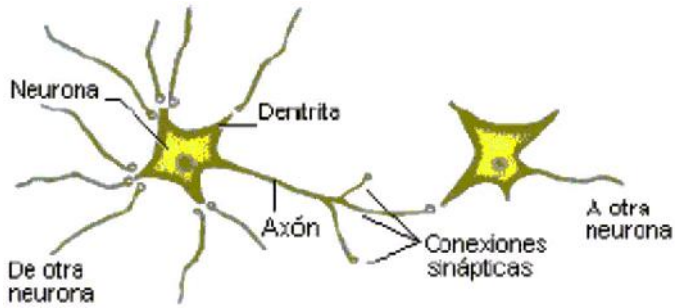


Figura 5.1: Neurona Biológica.

- Dendrítas. Conjunto de prolongaciones que reciben información de otras neuronas.

- Axón. Emite una señal como respuesta de la neurona, hacia otras neuronas.

Los axónes llegan hasta las dendritas de otras neuronas y establecen unas conexiones llamadas sinápsis, mediante las cuales se produce la comunicación entre neuronas, la cual puede ser eléctrica o química, dependiendo del espacio entre el axón de una y la dendrita de la otra. Las neuronas mandan mensajes que pueden tener un efecto sobre la receptora, ya sea de excitación o de inhibición. Todas las señales excitatorias e inhibitorias recibidas por la i-ésima neurona se combinan y en base al total de estimulación recibida, la neurona toma un valor de activación, que se traduce en la creación de nuevos impulsos nerviosos que se propagan a lo largo de su respectivo axón, hacia las neuronas con las cuales está conectada.

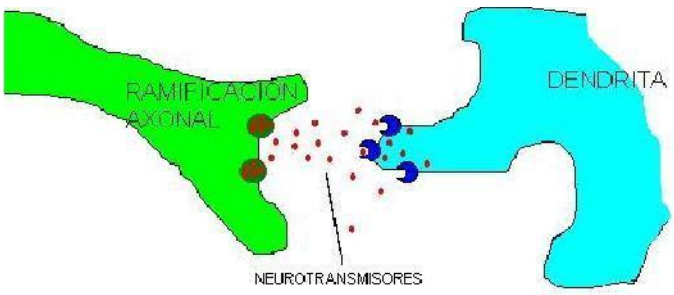


Figura 5.2: Estimulación biológica de una neurona.

Tanto las neuronas artificiales como las biológicas aprenden mediante ejemplos.

Aprender en sistemas biológicos implica ajustes en las conexiones sinápticas que existen entre las neuronas. Esto también lo hace una **RNA**. Una **RNA** es configurada para una aplicación especifica, tal como la clasificación de datos, a través del proceso de aprendizaje.

Hay muchas variantes de **RNA** pero en general, es una colección de procesadores de señales, conectados entre ellos de manera específico. Esquemáticamente, una **RNA** se representa en la forma de un grafo dirigido, donde los nodos simbolizan los elementos del procesador, los arcos las conexiones de modulación y las puntas de los arcos la dirección del flujo de la señal. Los elementos del procesador se agrupan generalmente juntos en estructuras conocidas como capas, donde cada elemento del procesador, en cada capa, realiza una integración de sus entradas para determinar el valor de activación.

El proceso comienza con la red entera en un estado pasivo. Un patrón externo se aplica a la capa de entrada, donde cada señal del patrón estimula uno de los elementos de la capa. Cada elemento en la capa genera una sola señal de salida, la magnitud de la salida esta en función del estímulo total recibido por el nodo. Las salidas producidas por todos los elementos de la capa se pasan a la capa subsecuente. Se repite este proceso, hasta que la última capa produce una salida para el patrón de entrada dado.

A continuación daremos una pequeña introducción sobre qué es y cómo funciona una neurona artificial, así como un conjunto de ellas, una red neuronal artificial, con la finalidad de dar un panorama de lo que son y como se comportan, enfocandonos en la arquitectura de red neuronal tipo *unidireccional*.

5.1.1. Neurona Artificial

La Figura 5.3 muestra la representación tradicional de una neurona artificial, donde se puede ver que ésta recibe un conjunto de valores de entrada, cada uno

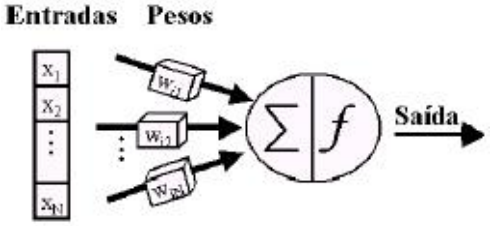


Figura 5.3: Neurona Artificial.

de los cuales esta asociado al valor de la conexión hacia la neurona (a este valor se le conoce como el peso de la conexión). La señal que conduce cada conexión es multiplicada por el valor del peso de la conexión, el resultado individual de la multiplicación, es sumado con los resultados de todas las entradas que tiene la neurona. El valor de la suma obtenida es conocido como el estado interno de la neurona y su salida es generada a través de la aplicación de una función, llamada función de activación. La función se aplica al estado interno de la neurona, que tiene como objetivo limitar el valor entre [-1, 1] ó [0, 1], dependiendo de la función elegida.

Practicamente existen dos pasos a seguir para el entrenamiento de la neurona:

1. Se calcula el valor del estado interno de la i-ésima neurona (también conocido como el valor de activación),

$$net_i = \sum_{j=1}^{N} w_{ij} \ * \ x_j, \tag{5.1}$$

2. Con el valor de activación ya conocido la i-ésima neurona produce una salida de acuerdo a la función de activación, f(·)

$$a_i = f(net_i), \tag{5.2}$$

Existen varias funciones de activación, para nuestros propositos hablaremos de la función *Log-Sigmoidal*

$$f(net) = \frac{1}{1 + e^{-net}}. \tag{5.3}$$

La *Log-Sigmoidal* produce una señal de salida que tiene dos estados y una región de transición. Es matemáticamente continua y además diferenciable, de acuerdo a la

formula (5.3). La Figura 5.4 muestra su representación grafica, esta función toma la entrada (que puede estar entre $\pm\infty$) y da como resultado un valor dentro del rango [0, 1].

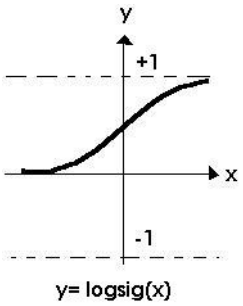


Figura 5.4: Función *Log-Sigmoidal*.

La derivada de la sigmoidal, ecuación (5.4), con respecto a la neurona de entrada es necesaria para calcular la señal del error en la neurona de salida.

$$f'(net) = \frac{\partial f(net)}{\partial net} = \frac{e^{-net}}{(1 + e^{-net})^2} \tag{5.4}$$

5.1.2. Red Neuronal Artificial

El vector X representa la capa de entrada, donde se puede ver como una capa *sorda* lo cual significa que solo pasa los valores tal cual a cada neurona de la siguiente capa, en este caso a la primer capa intermedia.

Las conexiones entre las neuronas se denominan pesos los cuales los representamos como w_{ij}, donde el subíndice i representa la neurona a la cual van a entrar los pesos de la capa en cuestión, subíndice j representa el elemento j-ésimo de la capa anterior.

La Figura 5.5, muestra una red con tres neuronas en cada una de las dos capas intermedias y una neurona en la capa de salida, donde podemos ver lo descrito anteriormente.

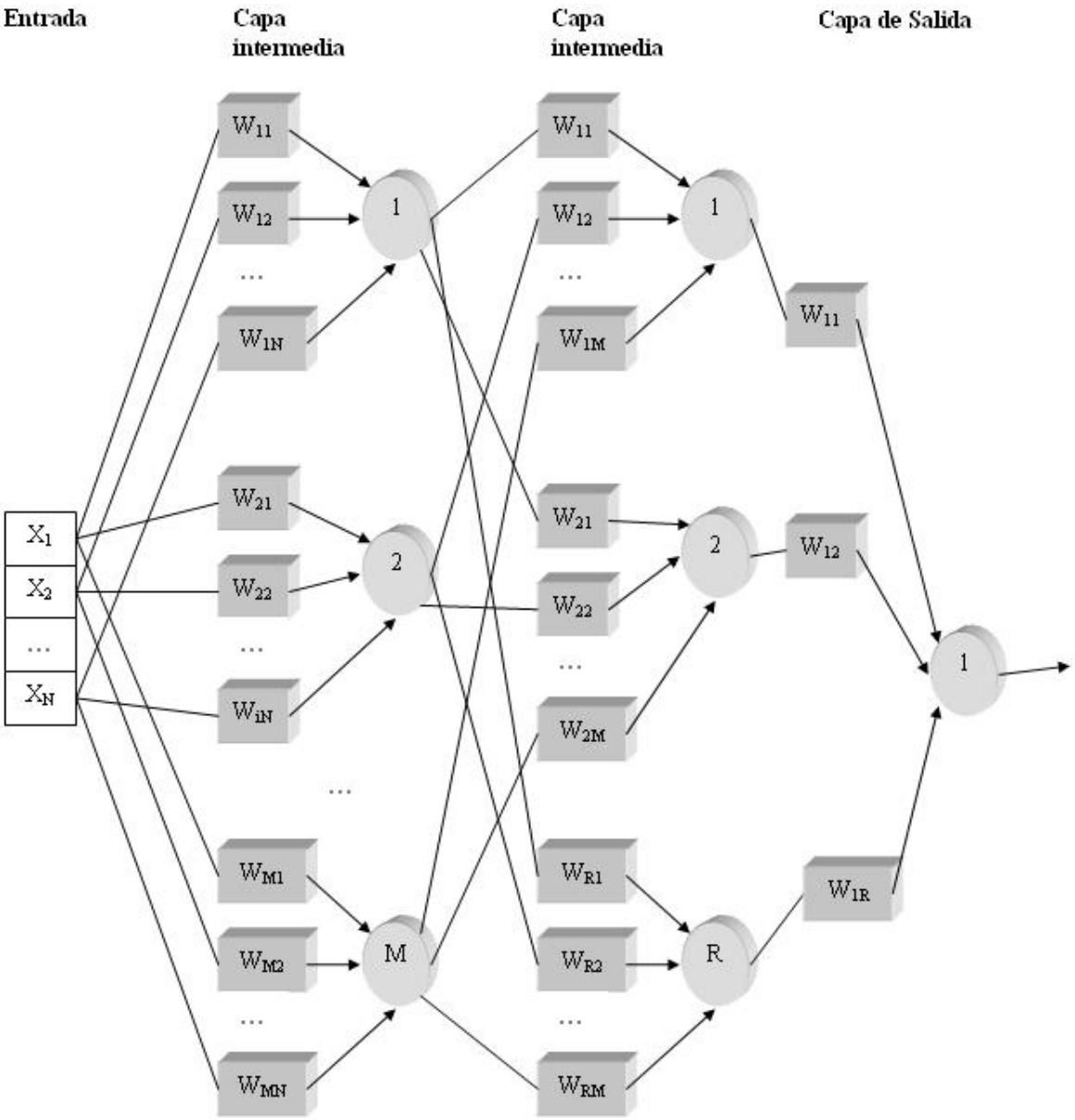


Figura 5.5: Ejemplo de una Red Neuronal Artificial.

Este tipo de red *unidireccional*, está conformada por un conjunto de neuronas distribuidas a lo largo de toda la red por medio de una capa de entrada, uno ó más capas intermedias y una capa de salida. La Figura 5.5 muestra una representación gráfica de este tipo de redes. Cada neurona tiene un único camino para comunicarse con las neuronas de la siguiente capa. La salida de una neurona es una entrada para las neuronas de la siguiente capa a la cual está conectada.

Si examinamos la configuración de la red la cual está basada en los valores de los pesos de la conexión, observamos los siguientes tres aspectos interesantes sobre el comportamiento de la red.

- Actua como una capa *sorda*, en otras palabras solo pasa el conjunto de patrones dados como ejemplos.
- La capa intermedia, actúa como un detector de características, su activación está en relación solamente con la respuesta de la capa anterior. Puede existir un número mayor o igual a cero capas intermedias, también conocidas como capas ocultas.
- La capa de salida, en este caso, consiste de una sola neurona, es ejecutada según las caracteristicas detectadas por la(s) capa(s) intermedia(s).

Durante la fase de entrenamiento, los pesos de la red se adaptan para reflejar el dominio del problema, mientras que en la fase de prueba, los pesos han sido congelados para cuando se le presente datos nuevos la red proporcione una clasificación para ellos.

Cabe señalar que los pesos son seleccionados al azar inicialmente, por la red, para evitar imponer cualquier juicio propio sobre el uso de la red. Donde $0 < w_{ij} < 1$.

Las redes son entrenadas por medio de un algoritmo de aprendizaje, así como existen diferentes funciones de activación para las neuronas, también existen varios algoritmos de aprendizaje para las redes.

5.2. Perceptrón Multi-Capa

PMC es entrenado con el algoritmo estándar de Back-Propagation (retropropagación de errores). Provee un método efectivo para evaluar el vector gradiente descendiente para una función de costo definida como:

$$\sum_{i=1}^{P}\sum_{j=1}^{N_L}(y_{j,i} - d_{j,i})^2 \tag{5.5}$$

donde:

P es el número de patrones de entrenamiento,

N_L es el número de nodos de la capa de salida L,

$y_{j,i}$ representa la salida correspondiente al nodo j para el patrón i,

$d_{j,i}$ representa la salida deseada correspondiente al nodo j para el patrón i.

Back-Propagation aprende a generar un mapeo a partir de patrones de entrada hacia los patrones de salida, por la minimización del error entre la salida actual producida por la red y la salida designada a través del conjunto de patrones dados como ejemplos. El proceso de aprendizaje comienza con la presentación de un patrón de entrada para Back-Propagation. El patrón de entrada es propagado a través de toda la red, dentro de la cual se produce el patrón de salida correspondiente. Entonces Back-Propagation determina el error para el patrón actual. Finalmente, cada nodo modifica los pesos de sus conexiones de entrada de modo que reduce significativamente la señal del error, esté proceso se repite para cada uno de los patrones de entrada.

El algoritmo de Back-Propagation, comienza creando una estructura con una configuración interna aleatoria para los pesos. Los patrones destinados para el entrenamiento son parejas $(x_i,\ y_i)\ \forall\ i = 1,\ \cdots,\ N$ donde cada x_i representa un vector de entrada y y_i su respuesta correspondiente. El proceso de entrenamiento de la red se describe a continuación (el siguiente listado se tomo de [2], página 31)

1. Tomar uno de los patrones destinados para el entrenamiento, $(x,\ y)$.

2. Utilizar el vector de entrada x, como la salida de la capa de la entrada de la neurona.

3. Calcular el valor de activación de cada neurona, usando la formula (5.1).

4. Aplique la función de activación.

5. Repetir el paso 3 y 4 para cada neurona dentro de todas las capas de la red.

6. Calcular el error para la capa de salida, para el patrón p a través de todas las k-ésimas neuronas de la capa de salida. El superíndice o, indica que se esta hablando sobre la capa de salida.

$$E^o_{pk} = (y_{pk} - x_k) * f'(net^o_k) \tag{5.6}$$

7. Calcular una estimación del error, para todas las neuronas de la capa oculta j. El superíndice h, indica que se esta hablando sobre la capa oculta.

$$E^h_{pj} = f'(net^h_j) * (\sum_{k=1}^{k} E^o_{pk}\ *\ w_{kj}) \tag{5.7}$$

8. Actualizar los valores de los pesos para las capas ocultas.

$$w_{ij}(t+1) = w_{ij}(t) + n(E^h_{pj})(x_i) \tag{5.8}$$

   donde n es un valor pequeño usado para limitar la cantidad de cambio permitido a cualquier conexión durante el ciclo de entrenamiento del patrón en cuestión.

9. Actualizar los valores de los pesos para la capa de salida.

$$w_{kj}(t+1) = w_{kj}(t) + n(E^o_{pk}) f(net^h_j) \tag{5.9}$$

10. Repetir los pasos 2 a 9 para todos los patrones del conjunto de datos destinados para el entrenamiento. A esto se le llama una época de entrenamiento.

11. Realizar los pasos 1 a 10 tantas veces como épocas sean necesarias para reducir el error a un valor mínimo. El cálculo del error se realiza para las neuronas de la capa de salida unicamente, a través de todos los patrones del entrenamiento.

$$Error = \sum_{p=1}^{P} \sum_{k=1}^{K} (E_{pk}^{o})^2 \tag{5.10}$$

5.3. Algunos ejemplos

Los siguientes ejemplos son implementaciones de **RNA** usando el tipo de red *unidireccional* **PMC** bajo **R**, la descripción de los conjuntos de datos se pueden ver en el Apéndice A. Hasta el momento no hay un criterio establecido para determinar la configuración de la red y esto depende más bien de la experiencia del diseñador de la red, en nuestro caso jugamos con el número de neuronas hasta obtener resultados aceptables, algunos de ellos los comparamos con resultados publicados en artículos para establecer si son aceptables o no.

Ejemplo 1 Los datos corresponden a los datos mostrados en el Apéndice A.1.

Cuadro 5.1: Pruebas con *M*.

	Entrenamiento	Prueba	Total
Clase 1	400	100	500 = 65.1 %
Clase 2	214	54	268 = 34.9 %
Total	614	154	768 = 100 %

El cuadro 5.1 muestra la distribución de los datos en dos archivos uno para el entrenamiento de la red y un segundo archivo para probarla. Los datos fueron normalizados usando la desviación estandar.

Recordemos que la desviación estandar mide la disperción de los datos, mientras

que la media nos dice donde se concentran los datos.

Cuadro 5.2: Resultados; Indios Pimas.

Predicción Vs Verdad	Diabetes	No Diabetes
Diabetes	78	22
No diabetes	19	35

En el cuadro 5.2 se pueden observar los resultados obtenidos con un error de predicción de 26.623 % de la red, la cual cuenta con ocho neuronas en la capa intermedia, ocho en la capa de entrada y una en la capa de salida. Éste cuadro se lee como: 78 patrones de datos fueron predecidos correctamente como la clase diabetes, mietras que 22 patrones se predijeron de la clase diebetes y en realidad pertenecen a la clase de no diabetes, por otro lado 19 patrones fueron predecidos de la clase no diabetes cuando en realidad pertenecen a la clase diabetes, mietras que 35 patrones se predijeron correctamente como de la clase no diabetes.

Ejemplo 2 Los datos corresponden a los datos mostrados en el Apéndice A.2. El cuadro 5.3 muestra la distribución de los datos en dos archivos uno para el

Cuadro 5.3: Distribución de los datos; Iris.

	Entrenamiento	Prueba	Total
Clase 1	25	25	50 = 33.3 %
Clase 2	25	25	50 = 33.3 %
Clase 3	25	25	50 = 33.3 %
Total	75	75	150 = 100 %

entrenamiento de la red y un segundo archivo para probarla.

En el cuadro 5.4 se pueden observar los resultados obtenidos con un error de

Cuadro 5.4: Resultados; Iris.

Predicción Vs Verdad	Setosa	Versicolor	Virginica
Setosa	24	0	1
Versicolor	0	25	0
Virginica	0	0	25

predicción de 1.33 % de la red ante el archivo de prueba. La red esta formada por 2 neuronas en la capa intermedia, 4 en la capa de entrada y 1 en la capa de salida. Este cuadro se lee muy parecido al cuadro 5.2 solo que ahora se esta hablando de tres clases diferentes. El cuadro 5.4 se lee como: 24 patrones fueron predecidos de la clase setosa correctamente, mietras que un dato fué predecido de la clase setosa cuando en realidad pertenece a la clase virginica, por otro lado los 25 patrones de la clase versicolor se predijeron correctamente, así como también se predijeron bien los 25 patrones correspondientes a la clase virginica.

5.4. Variables Importantes

Este método de caja negra no cuenta con un mecanismo inherente para dar una aproximación sobre cuáles variables son importantes y cuáles no, como en **BA**, para tratar de contestar las preguntas de ¿Cómo saber cuáles de las entradas son realmente necesarias?, ¿Se necesitan de todas las entradas para poder clasificar algo adecuadamente? hemos permutado manualmente cada una de las entradas, los resultados obtenidos serán utilizados dentro de un ejemplo en el capítulo 2, las primeras cuatro columnas del cuadro 5.5, corresponden a cada uno de los atributos de entrada a la red, un uno representa que los datos correspondientes a ese atributo han sido permutados aleatoriamente cinco veces, la quinta columna representa el promedio del error de predicción, para aportar más detalles también se agregaron las columnas correspondientes a la desviación estandar, el mínimo y el máximo valor obtenido según las cinco ejecuciones con las respectivas permutaciones.

Cuadro 5.5: Permutaciones; Iris.

Permutaciones				Error de predicción	Desviación Estandar	Mínimo	Máximo
0	0	0	0	1.33 %	0.00	1.33	1.33
1	0	0	0	2.93 %	1.11	1.33	4.00
0	1	0	0	2.13 %	0.73	1.33	2.66
1	1	0	0	6.13 %	1.52	4.00	8.00
0	0	1	0	58.06 %	2.45	56.00	61.33
1	0	1	0	57.60 %	4.25	52.00	62.66
0	1	1	0	59.20 %	3.21	56.00	62.66
1	1	1	0	60.00 %	4.10	56.00	65.33
0	0	0	1	22.13 %	1.19	21.33	24.00
1	0	0	1	22.93 %	3.69	18.66	26.66
0	1	0	1	25.06 %	3.93	20.00	30.66
1	1	0	1	21.33 %	4.42	14.66	26.66
0	0	1	1	66.66 %	4.10	61.33	72.00
1	0	1	1	69.33 %	6.03	62.66	76.00
0	1	1	1	67.20 %	5.04	60.00	72.00
1	1	1	1	66.93 %	3.45	62.66	72.00

Según los resultados observados en el cuadro 5.5 podemos intuir que la variable que más puede afectar la predicción es la tercera y que adicionalmente tiene correlación con la cuarta entrada, la variable que menos afecta el resultado de la predicción es la segunda entrada aparentemente.

Nota: La función bajo **R** para realizar el entrenamiento de una red neuronal artificial es la siguiente.

```
nnet(x, y, size, data, subset, Wts, rang= 0.7, decay= 0, maxit= 100, ...)
```

donde,

formula: Una formula de la forma: $clases \tilde{} x_1 + x_2 + ...$

x: Matriz con los valores para el entrenamiento.

y: Matriz con las respuestas correspondientes a los patrones de entrenamiento.

size: Número de neuronas para la capa oculta.

data: Conjunto de datos de el cual las variables especificadas en *formula* deben tomar sus valores.

subset: Un vector de indices que especifica los casos que se utilizarán en el entrenamiento.

Wts: Pesos iniciales. Si no se proporcionan entonces serán seleccionados aleatoriamente.

rang: Pesos al azar iniciales en [-rang, rang]. Valor cerca de 0.5 a menos que las entradas sean grandes, en tal caso elegirlo para rang * max($|x|$) cerca de uno.

decay: Parámetro para el decaimiento del peso.

maxit: Máximo número de iteraciones. Por omisión 100.

...: Indica la posibilidad de agregar más opciones.

Capítulo 6

Aplicación de CSR para radiografiar clasificadores tipo caja negra

La pregunta central en este capítulo es como radiografiar un clasificador basado en un método de tipo caja negra para entender (mejor) la relación entre las variables predictoras y su predicción, tilizando el **Modelo CSR**.

La pregunta de como entender la relación entre variables predictoras y de respuesta para un método de tipo caja negra, tiene una mayor tradición en el campo de redes neuronales.

Un método muy sencillo es elegir un predictor, permutar los valores observados para este predictor y medir el deterioro de la calidad en la clasificación con este conjunto alterado. Leo Breiman ha promovido mucho este enfoque en el contexto de bosques al azar. Una limitación fuerte es que se caracteriza la influencia de un solo predictor a la vez.

Uno de los caminos que se quiere explorar en este capítulo es extender el método de Breiman a permutaciones de varias variables de entrada al mismo tiempo. Es-

to conducirá a un nuevo conjunto de datos con variables predictoras binarias que indican cuales entradas fueron permutadas y como respuesta la perdida de calidad correspondiente del clasificador.

6.1. Prueba con Bosques al Azar

Recordemos que un **BA** (Capítulo 4) es un conjunto de árboles de clasificación donde cada árbol trabaja con solo una parte del total de datos destinados para realizar la construcción del bosque. Como ya se mencionó este método cuenta internamente con la implementación de una sola permutación a la vez.

Ejemplo (Prueba) 1 *Usando el archivo de datos: Plantas Iris; la descripción de este conjunto de datos puede ser consultado en el Apéndice A.2.*

El entrenamiento de **BA** reportó un error de clasificación de 0.093 % y un error en el conjunto de prueba de 0.013 %. Para el entrenamiento se utilizaron los valores por omisión de $ntree = 500$ y de $mtry = 2$. La Figura 6.1 muestra un conjunto de cinco gráficas con información sobre como se comportan las variables, según **BA**. En estas gráficas se puede observar que la variable más importante es X_4 y la menos importante es la variable X_2.

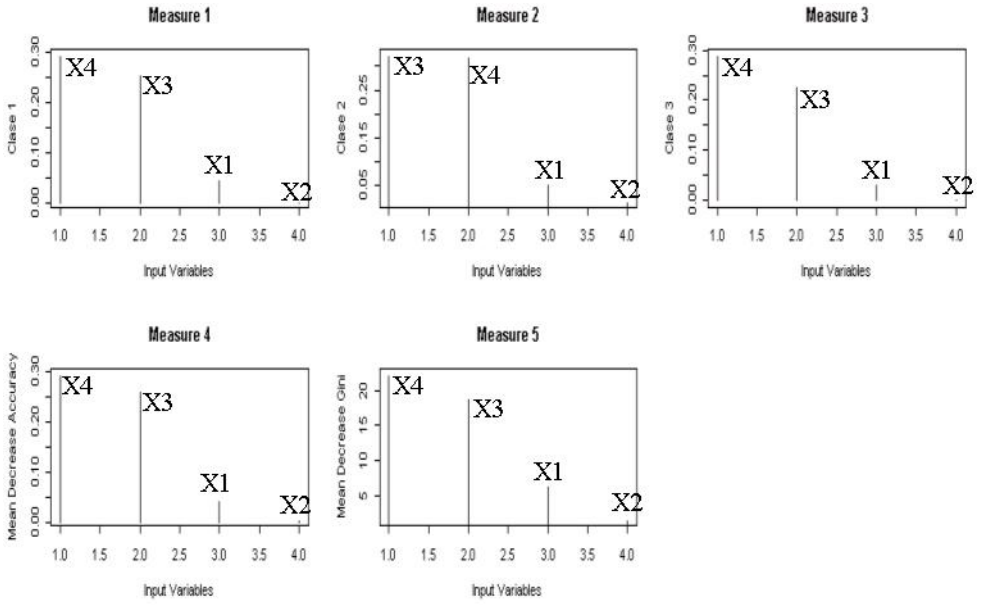


Figura 6.1: Resultado obtenido de **BA**.

Para extender lo anterior, se realizaron una serie de permutaciones para cada variable de entrada, con el fin de ver el deterioro de la calidad en la predicción del bosque. El proceso con el cual se realizaron las permutaciones a las variables de entrada y verificar su calidad es el siguiente:

```
n= 4
combinacion= 0, 0, 0, 0
             1, 0, 0, 0
             0, 1, 0, 0
                 ...
             0, 1, 1, 1
             1, 1, 1, 1
Para 1 <= i <= 2^n
     Para 1 <= j <= 5
          Para todo numero uno en la i-esima combinacion
               permutar aleatoriamente la entrada
               correspondiente a los unos en el archivo
               destinado para la prueba
```

```
14            fin para
15            Y_Observada= predict(objeto random forest ,
16                                 archivo prueba)
17            Error_prediccion= Calcular Y_Esperada menos
18                                 Y_Observada
19        fin para
20  fin para
```

La función *predict* de **R** tiene la siguiente estructura:

```
predict(objeto, nuevos datos)
```

donde,

objeto: Debe de ser un objeto del tipo *randomforest*.

nuevos datos: Matriz de datos que se quieren probar.

Como se especifica en el pseudo código anterior, se realizaron cinco iteraciones de cada permutación, el resultado que se reporta a continuación es el promedio del error de predicción para cada una de las permutaciones. El cuadro 6.1 muestra los resultados dados por **BA**.

La primera columna muestra el resultado del promedio del error de predicción del archivo de prueba, en las siguientes cuatro columnas se puede observar la codificación que se utilizó para representar cuales columnas fueron permutadas. Cabe aclarar que un número uno, en estas últimas cuatro columnas, indica que esa variable fue permutada para obtener el valor mostrado en la primera columna.

El cuadro 6.2 muestra un conjunto de modelos probados y los resultados obtenidos para el caso de orden dos.

Los resultados mostrados en el cuadro 6.2 al fijarnos en la columna correspondiente al valor de R^2, sugieren que no se pueden quitar las dos últimas variables ya

Cuadro 6.1: Permutaciones para **BA**.

Error de predicción	X_1	X_2	X_3	X_4
0.013	0	0	0	0
0.013	1	0	0	0
0.013	0	1	0	0
0.016	1	1	0	0
0.152	0	0	1	0
0.267	1	0	1	0
0.187	0	1	1	0
0.283	1	1	1	0
0.280	0	0	0	1
0.379	1	0	0	1
0.304	0	1	0	1
0.397	1	1	0	1
0.587	0	0	1	1
0.648	1	0	1	1
0.549	0	1	1	1
0.597	1	1	1	1

Cuadro 6.2: Resultados del **Modelo CSR**.

Modelo	Caso de orden dos		
	gl	R^2	Multiple R-Squared
{0, 0, 0, 0}	5	0.969	0.989
{*, 0, 0, 0}	9	0.942	0.965
{0, *, 0, 0}	9	0.980	0.988
{0, 0, *, 0}	9	0.485	0.691
{0, 0, 0, *}	9	0	0.318

que al hacerlo la calidad del modelo es bastante mala. Por el contrario al quitar alguna de las dos primeras variables la calidad de los resultados mostrados son buenos. El cuadro 6.3 muestra algunas pruebas realizadas con el fin de determinar si es posible eliminar del modelo las interacciones con las dos primeras variables.

Cuadro 6.3: Resultados del **Modelo CSR**.

Modelo	Caso de orden dos		
	gl	R^2	Multiple R-Squared
{*, *, 0, 0}	12	0.954	0.963
{*, *, 0, 3}	13	0.520	0.584
{*, *, 0,-3}	13	0.622	0.673
{*, *, 4, 0}	13	0.753	0.785
{*, *,-4, 0}	13	0.821	0.845
{*, *, 4, 3}	14	0.535	0.566
{*, *,-4, 3}	14	0	0.016
{*, *, 4,-3}	14	0	0.036
{*, *,-4,-3}	14	0.641	0.665

Del cuadro 6.3 podemos ver que el mejor modelo es {*, *, 0, 0} con una calidad casi perfecta ya que el tiene la mejor R^2, sin embargo de las combinaciones restantes los mejores modelos son {*, *, 4, 0} y {*, *, -4, 0}. La Figura 6.3 muestra las gráficas correspondientes a los residuos de los modelos estimados utilizando los modelos que reportaron una mejor calidad utilizando alguna interacción.

Del cuadro 6.3 es normal suponer que el mejor modelo es {*, *, -4, 0}, sin embargo de las gráficas mostradas en la Figura 6.3 que muestran los cuantiles de los residuos vemos que en realidad el mejor es {*, *, 4, 0}, ya que de los dos grafos es el que mejor se ajusta a una recta y se cumple con el supuesto de normalidad.

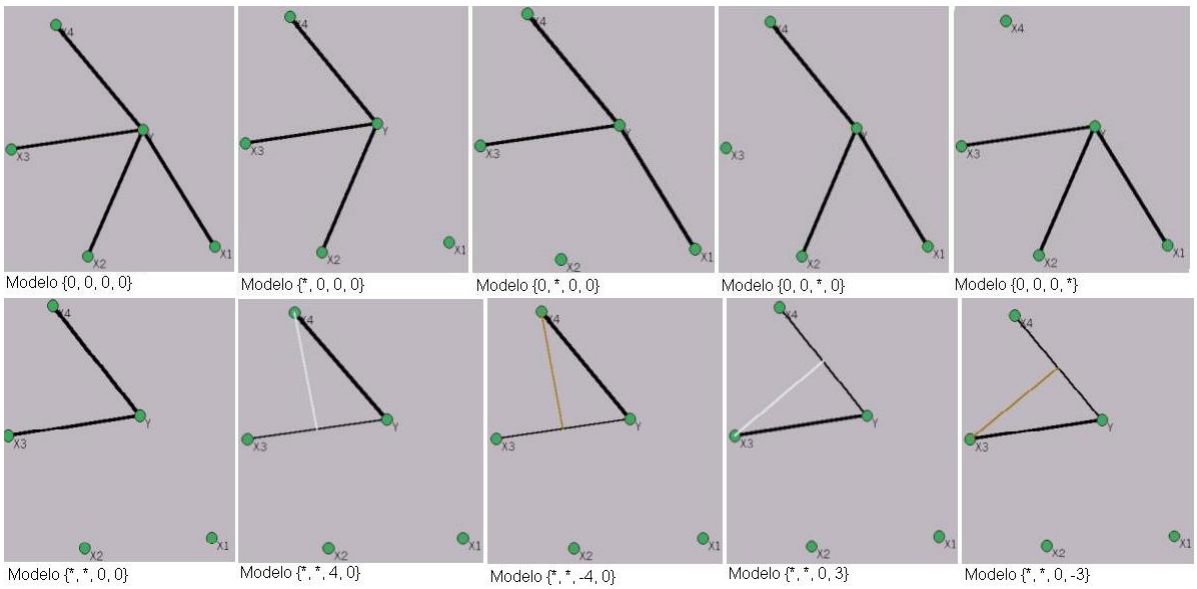


Figura 6.2: Gráficas de modelos.

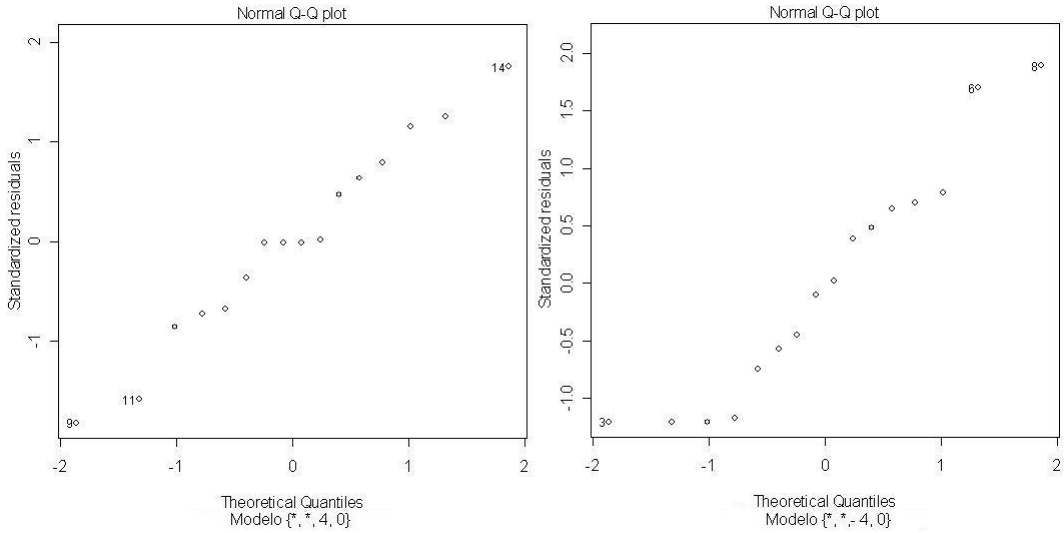


Figura 6.3: Grafos de cuantiles de los residuos para modelos con **RF**.

6.2. Prueba con Redes Neuronales Artificiales

Una **RNA** (Capítulo 5) en general, es una colección de procesadores de señales, conectados a través de conexiones. Esquemáticamente, una **RNA** se representa en la forma de un grafo dirigido, donde los nodos simbolizan los elementos del procesador, los arcos las conexiones de modulación y las punta de los arcos la dirección del flujo de la señal.

Ejemplo (Prueba) 2 *Usando el archivo de datos: Plantas Iris; la descripción de este conjunto de datos puede ser consultado en el Apéndice A.2.*

El entrenamiento de **RNA** reportó un error de clasificación de 0.0495 % y un error en el conjunto de prueba de 0.013 %. Para el entrenamiento se utilizaron los valores $size = 2$, $rang = 0.1$, $decay = 5e - 04$, $maxit = 500$.

El seudo código mostrado en la sección anterior para la realización de las permutaciones, es el mismo proceso por el cual se obtuvo el archivo mostrado en el cuadro 6.4.

El cuadro 6.5 muestra un conjunto de modelos probados y los resultados obtenidos para los dos casos: caso de orden dos y el caso general.

Los resultados mostrados en el cuadro 6.5, sugieren que no se puede quitar la tercera variable ya que al hacerlo la calidad del modelo es bastante mala. Por el contrario al quitar alguna de las variables restantes la calidad de los resultados mostrados son buenos. El cuadro 6.6 muestra las pruebas realizadas con el fin de determinar si es posible eliminar del modelo las interacciones con la tercera variable.

Del cuadro podemos ver que el mejor modelo es {*, *, 0, 0} con una calidad casi perfecta, sin embargo de las combinaciones restantes los mejores modelos son {*, *, 0, 3} y {*, *, 0, -3}. La Figura 6.4 muestra las gráficas correspondientes a los cuantiles de los residuos, utilizando los modelos que reportaron una mejor calidad,

Cuadro 6.4: Permutaciones para **RNA**.

Error de predicción	X_1	X_2	X_3	X_4
0.013	0	0	0	0
0.053	1	0	0	0
0.040	0	1	0	0
0.075	1	1	0	0
0.648	0	0	1	0
0.589	1	0	1	0
0.613	0	1	1	0
0.587	1	1	1	0
0.216	0	0	0	1
0.173	1	0	0	1
0.205	0	1	0	1
0.219	1	1	0	1
0.709	0	0	1	1
0.637	1	0	1	1
0.699	0	1	1	1
0.669	1	1	1	1

Cuadro 6.5: Resultados de **Modelo CSR**.

Modelo	Caso de orden dos		
	gl	R^2	Multiple R-Squared
{0, 0, 0, 0}	5	0.996	0.998
{*, 0, 0, 0}	9	0.988	0.993
{0, *, 0, 0}	9	0.995	0.997
{0, 0, *, 0}	9	0	0.048
{0, 0, 0, *}	9	0.910	0.946

Cuadro 6.6: Resultados de **Modelo CSR**.

Modelo	Caso de orden dos		
	gl	R^2	Multiple R-Squared
{ *, *, 0, *}	14	0.936	0.940
{ *, 0, 0, *}	12	0.926	0.941
{ *, 3, 0, *}	13	0.932	0.941
{ *,-3, 0, *}	13	0.931	0.940
{ *, 0, 2, *}	13	0.415	0.493
{ *, 0,-2, *}	13	0.363	0.448
{ *, 3, 2, *}	14	0.292	0.339
{ *,-3, 2, *}	14	0.269	0.318
{ *, 3,-2, *}	14	0.237	0.288
{ *,-3,-2, *}	14	0.259	0.308
{ *, *, 0, 0}	12	0.990	0.992
{ *, *, 0, 3}	13	**0.981**	0.983
{ *, *, 0,-3}	13	**0.940**	0.948
{ *, *, 4, 0}	13	0.537	0.598
{ *, *,-4, 0}	13	0.351	0.438
{ *, *, 4, 3}	14	0.499	0.533
{ *, *, 4,-3}	14	0.180	0.235
{ *, *,-4, 3}	14	0.091	0.151
{ *, *,-4,-3}	14	0.359	0.402
{ 0, *, 0, *}	12	0.930	0.944
{ 0, *, 1, *}	13	0.452	0.525
{ 0, *,-1, *}	13	0.330	0.420
{ 3, *, 0, *}	13	0.931	0.940
{-3, *, 0, *}	13	0.935	0.944
{ 3, *, 1, *}	14	0.279	0.327
{ 3, *,-1, *}	14	0.249	0.299
{-3, *, 1, *}	14	0.327	0.372
{-3, *,-1, *}	14	0.206	0.259

utilizando alguna interacción.

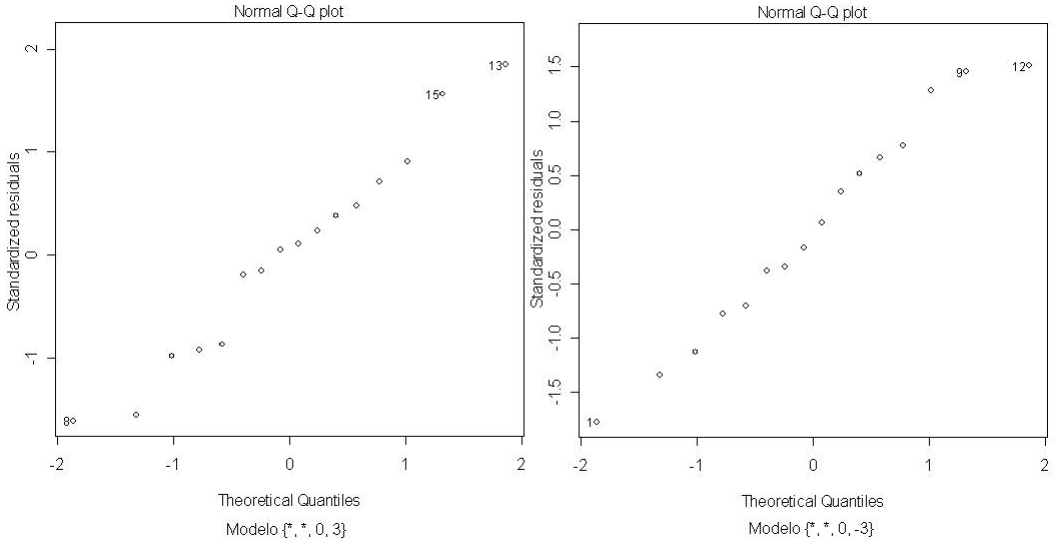


Figura 6.4: Q-Q Modelos con **RNA**.

Del cuadro 6.6 es normal suponer que el mejor modelo es {*, *, 0, 3}, sin embargo de las gráficas 6.4 vemos que en realidad el mejor es {*, *, 0, -3}, ya que de los dos grafos es el que mejor se ajusta a una recta, entonces no se violan las suposiciones del modelo.

En el Apéndice B se explica la prueba gráfica de normalidad, donde podemos ver como se realizan y que significan las gráficas de los cuadros 6.3 y 6.4.

Conclusiones

Expondremos los objetivos que se han alcanzado, así como también las conclusiones a las que se han llegado durante el desarrollo del presente documento.

- Elaboración de un programa sobre la implementación del **Modelo CSR**, bajo el *Lenguaje de Programación Estadístico R*. Se elaboraron los métodos de gauss-jordan y un procedimiento para encontrar ciclos dentro de un cubo n-dimencional y por medio de este último verificar si estamos hablando de un modelo consistente o inconsistente.

- Descripción de modelos de tipo caja negra: *Bosques al Azar* y *Redes Neuronales Artificiales*. Utilizando la aplicación del **Modelo CSR** para ajustar el mejor modelo posible para el conjunto de datos con el cual se este trabajando.

  Elaboramos como los **Modelos CSR** pueden servir para entender mejor el funcionamiento de los clasificadores tipo caja negra. De esta menera, entender los resultados reportados por Breiman. Cabe señalar que existen conjuntos de datos para los cuales no fue posible ajustar un modelo para ellos, ya que el conjunto de datos no se presto para tal cosa.

- El ambiente de software utilizando el *Lenguaje de Programación Estadístico R* desarrollado para la implementación del método *Context Sensitive Regression Models*, ha presentado muy buenos resultados.

Apéndice A

Descripción de Datos

Dentro de este apartado se describe brevemente cada conjunto de datos con los cuales se trabajó en la realización de este proyecto, algunos de los cuales se pueden encontrar en la pagina web http://www.ics.uci.edu/ mlearn/databases/. consultada a lo largo de la realización de este documento, aquellos que no se pueden encontrar en dicha ruta electronica son proporcionados por medio de tablas.

A.1. Diabetes de indios pimas

Los datos corresponden a una población de Indios Pimas cerca de Phoenix, Arizona, E. U., la información es de mujeres mayores de 21 años, con el proposito de detectarles diabetes. Se cuenta con ocho atributos y una variable que indica si se presenta la enfermedad de la diabetes o no.

Descripción de los datos:

- Número de casos: 768
- Número de clases: 2
    - No tiene diabetes.
    - Si tiene diabetes.

- Distribución de las clases
  - Clase 1 con 500 datos (65.1 %).
  - Clase 2 con 268 datos (34.9 %).
- Número de atributos: 8 más uno de clasificación
  1. Número de veces embarazada.
  2. Concentración de la glucosa.
  3. Presión arterial alterada.
  4. Grueso del doble de la piel del triceps.
  5. Insulina.
  6. Índice de masa corporal.
  7. Función de diabetes.
  8. Edad.
  9. Clasificación (Diabetes, No diabetes).

A.2. Plantas Iris

El modelo contiene tres clases de 50 casos cada uno, donde cada clase se refiere a un tipo de planta.

Descripción de los datos:

- Número de casos: 150
- Número de clases: 3
  - Iris Setosa.
  - Iris Versicolour.
  - Iris Virginica.

- Distribución de las clases
  - Clase 1 con 50 datos (33.33 %).
  - Clase 2 con 50 datos (33.33 %).
  - Clase 3 con 50 datos (33.33 %).
- Número de atributos: 4 más uno de clasificación:
  1. Longitud del sepal en centrimetros.
  2. Ancho del sepal en centrimetros.
  3. Longitud del pétalo en centrimetros.
  4. Ancho del pétalo en centrimetros.
  5. Clasificación (Setosa, Versicolour ó Virginica).

A.3. Las mujeres y las matemáticas

Se refiere a un estudio entre 1190 estudiantes de la secundaria de New Yersey sobre su actitud hacia las matemáticas y el impacto de una serie de conferencias para animar el interés en esa área.

Descripción de los datos:

- Número de ejemplos: 32
- Y; Representa el hecho de si el estudiante necesitará matemáticas en el futuro.

  0 Conviene

  1 No conviene
- X_1; Representa el echo de si el estudiante pone atención a las lecturas.

  0 Si

  1 No

A. Descripción de Datos

- X_2; Representa el sexo del estudiante.

  0 Mujer

  1 Hombre

- X_3; Representa el tipo de escuela.

  0 Sub-urbana

  1 Urbana

- X_4; Representa los cursos que prefiere el estudiante.

  0 Matematicas

  1 Artes

- X_5; Representa los planes a futuro.

  0 Seguir estudiando

  1 Trabajar

En la siguiente tabla se presentan los datos descritos por si se deseará repetir alguno de los experimentos planteados usando estos datos. La primera columna representa aquellos alumnos que con las caracteristicas descritas por cada X_i con $i = \{1, \ldots, 5\}$ es conveniente que tomen cursos de matemáticas, en cambio la segunda columna indica que para el número de personas que aparece en la casilla y con las mismas características no es conveniente que tomen dichos cursos.

Cuadro A.1: Conjunto de datos de matemáticas.

Y= 0	Y= 1	X_1	X_2	X_3	X_4	X_5
37	16	0	0	0	0	0
27	11	1	0	0	0	0
51	10	0	1	0	0	0
48	19	1	1	0	0	0

continuación...						
Y= 0	Y= 1	X_1	X_2	X_3	X_4	X_5
51	24	0	0	1	0	0
55	28	1	0	1	0	0
109	21	0	1	1	0	0
86	25	1	1	1	0	0
16	12	0	0	0	1	0
15	24	1	0	0	1	0
7	13	0	1	0	1	0
6	7	1	1	0	1	0
32	55	0	0	1	1	0
34	39	1	0	1	1	0
30	26	0	1	1	1	0
31	19	1	1	1	1	0
10	9	0	0	0	0	1
8	4	1	0	0	0	1
12	8	0	1	0	0	1
15	9	1	1	0	0	1
2	8	0	0	1	0	1
1	9	1	0	1	0	1
9	4	0	1	1	0	1
5	5	1	1	1	0	1
7	8	0	0	0	1	1
10	4	1	0	0	1	1
7	6	0	1	0	1	1
3	4	1	1	0	1	1
5	10	0	0	1	1	1
2	9	1	0	1	1	1
1	3	0	1	1	1	1

continuación...						
Y= 0	Y= 1	X_1	X_2	X_3	X_4	X_5
3	6	1	1	1	1	1

A.4. Permutaciones datos Iris

Descripción de los datos:

Esta tabla de datos se utilizo para extraer el cuadro expuesto en la sección 5.4. Los datos corresponden a la ejecución de una red neuronal, descrita en el mismo apartado.

Cuadro A.2: Permutaciones; Iris.

Permutaciones				Error de predicción
0	0	0	0	1.33 %
0	0	0	0	1.33 %
0	0	0	0	1.33 %
0	0	0	0	1.33 %
0	0	0	0	1.33 %
1	0	0	0	2.66 %
1	0	0	0	1.33 %
1	0	0	0	2.66 %
1	0	0	0	4.00 %
1	0	0	0	4.00 %
0	1	0	0	1.33 %
0	1	0	0	2.66 %
0	1	0	0	1.33 %
0	1	0	0	2.66 %
0	1	0	0	2.66 %
1	1	0	0	5.33 %

continuación...				
Permutaciones				Error de predicción
1	1	0	0	4.00 %
1	1	0	0	6.66 %
1	1	0	0	6.66 %
1	1	0	0	8.00 %
0	0	1	0	56.00 %
0	0	1	0	57.33 %
0	0	1	0	61.33 %
0	0	1	0	60.00 %
0	0	1	0	56.00 %
1	0	1	0	52.00 %
1	0	1	0	58.66 %
1	0	1	0	54.66 %
1	0	1	0	62.66 %
1	0	1	0	60.00 %
0	1	1	0	62.66 %
0	1	1	0	62.66 %
0	1	1	0	57.33 %
0	1	1	0	56.00 %
0	1	1	0	57.33 %
1	1	1	0	56.00 %
1	1	1	0	60.00 %
1	1	1	0	62.66 %
1	1	1	0	56.00 %
1	1	1	0	65.33 %
0	0	0	1	24.00 %
0	0	0	1	21.33 %
0	0	0	1	21.33 %
0	0	0	1	21.33 %

continuación...				
Permutaciones				Error de predicción
0	0	0	1	22.66 %
1	0	0	1	18.66 %
1	0	0	1	26.66 %
1	0	0	1	26.66 %
1	0	0	1	22.66 %
1	0	0	1	20.00 %
0	1	0	1	30.66 %
0	1	0	1	24.00 %
0	1	0	1	26.66 %
0	1	0	1	24.00 %
0	1	0	1	20.00 %
1	1	0	1	26.66 %
1	1	0	1	14.66 %
1	1	0	1	22.66 %
1	1	0	1	22.66 %
1	1	0	1	20.00 %
0	0	1	1	61.33 %
0	0	1	1	72.00 %
0	0	1	1	68.00 %
0	0	1	1	68.00 %
0	0	1	1	64.00 %
1	0	1	1	74.66 %
1	0	1	1	64.00 %
1	0	1	1	62.66 %
1	0	1	1	76.00 %
1	0	1	1	69.33 %
0	1	1	1	60.00 %
0	1	1	1	64.00 %

continuación...				
Permutaciones				Error de predicción
0	1	1	1	72.00 %
0	1	1	1	70.66 %
0	1	1	1	69.33 %
1	1	1	1	66.66 %
1	1	1	1	68.00 %
1	1	1	1	62.66 %
1	1	1	1	72.00 %
1	1	1	1	65.33 %

A.5. Datos Artificiales

Como parte de las pruebas realizadas se utilizó un conjunto de datos artificial. El cual se creo de la siguiente manera:

Se crearón tres variables aleatorias utilizando le distribución *Bernoulli*. Consiste en realizar un experimento aleatorio una sóla vez y observar si cierto suceso ocurre o no, siendo p la probabilidad de que esto sea así (éxito) y $1 - p$ el que no lo sea (fracaso). Esta distribución se denota por $X \tilde{} Ber(p)$. También se crearon dos variables más con la distribución *Normal*. La distribución de una variable normal está determinada por dos parámetros: su media (μ) y su desviación estándar (δ).

Lo descrito anteriormente lo podemos resumir como:

$$
\begin{aligned}
&X_1 \tilde{} Ber(p = 0{,}5)\\
&X_2 \tilde{} Ber(p = 0{,}5)\\
&X_3 \tilde{} Ber(p = 0{,}5)\\
&X_4 \tilde{} N(\mu = 0, \sigma^2 = 1)\\
&X_5 \tilde{} N(\mu = 0, \sigma^2 = 1)\\
&\Theta = \tfrac{1}{1+e^{-X_1 - X_3 - (X_2 * X_3)}}\\
&Y \tilde{} Ber(p = \Theta).
\end{aligned}
$$

A.6. Datos Sonar

Estos datos resultaron del sonido de una bala contra una roca y un cilindro metálico. El propósito es predecir si el eco viene de roca o de metal.

Descripción de los datos

- Número de ejmplos: 207
- Número de clases: 2
    1. Roca
    2. Metal
- Número de atributos: 30 más uno de clasificación

Apéndice B

Prueba gráfica de normalidad

Dentro de este apéndice se explica la prueba gráfica de normalidad. Comó se crean y para que sirven.

Distribución Normal (o distribución gausseana)

La distribución de una variable normal está completamente determinada por dos parámetros, su media y su desviación estándar, denotadas generalmente por μ y σ respectivamente. Con esta notación, la densidad de la normal viene dada por la ecuación:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \ -\infty \ < \ x \ < \ \infty \tag{B.1}$$

que determina la curva en forma de campana, que tambien conocida como la campana de *Gauss*. Así, se dice que una característica X sigue una distribución normal de media μ y varianza σ^2, y se denota como $X \ \approx \ N(\mu, \sigma)$, si su función de densidad viene dada por la ecuación B.1. La Figura B.1 muestra la representación común de una distribución normal con media uno y desviación estándar cero.

Histogramas

Estos tipos de gráficas se utilizan para representar distribuciones de frecuencias. Por ejemplo ver Figura B.2 donde se muestra un ejemplo de un histograma.

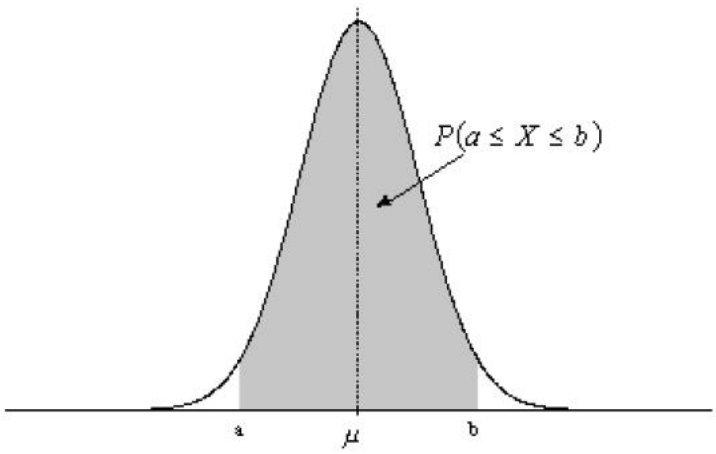


Figura B.1: Campana de Gauss.

Donde existe un rectangulo vertical por cada caracteristica de los datos, la altura representa el número de observaciones de esa caracteristica en particular.

Gráficos de probabilidad normal

Es una herramienta gráfica para comprobar si un conjunto de datos puede considerarse o no procedentes de una distribución normal. La idea básica consiste en enfrentar, en un mismo gráfico, los datos que han sido observados frente a los datos teóricos que se obtendrían con una distribución gaussiana.

Una curva en forma de U o con alguna curvatura, como en el caso de la edad en la Figura B.2, significa que la distribución es asimétrica con respecto a la gaussiana, mientras que un gráfico en forma de S significará que la distribución tiene colas mayores o menores que la normal, esto es, que existen pocas o demasiadas observaciones en las colas de la distribución.

Como en las graficas de la Figura B.2 se sobreponen dos graficas, el histograma de frecuencias a la cual se le trata de ajustar una campana de gauss, sin embargo si lo hacemos así no obtendremos justo la normal que pasa sobre el histograma, entonces lo que se hace es, verificar por medio de una gráfica Q-Q donde, los puntos donde cada rectangulo de nuestro histograma representan los cuantiles observados

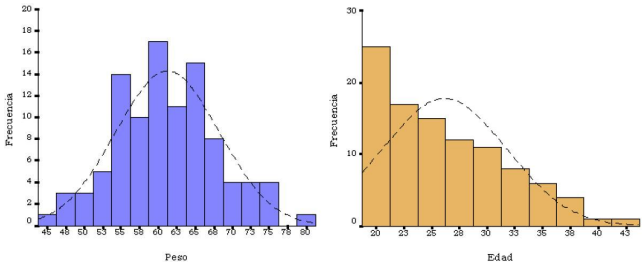


Figura B.2: Campana de gauss e histogramas.

los cuales los vamos a graficar uno a uno contra la frecuencia real del rectangulo en cuestión, al cual se le denomina cuantil. Es por esta razón que se llaman gráficas Q-Q (cuantil contra cuantil), el resultado será una gráfica con puntos uno por observación del histograma, si estos puntos forman una linea recta entonces obtenemos una campana de gauss que se ajusta al histograma, lo cual quiere decir que los datos vienen de una distribución normal.

Más formalmente tenemos: Si la distribución de la variable coincide con la normal, los puntos se concentrarán en torno a una línea recta, aunque conviene tener en cuenta que siempre tenderá a observarse mayor variabilidad en los extremos. En los gráficos Q-Q se confrontan las proporciones acumuladas de una variable con las de una distribución normal. Los gráficos Q-Q se obtienen de modo análogo, esta vez representando los cuantiles respecto a los cuantiles de la distribución normal. Además de permitir valorar la desviación de la normalidad, los gráficos de probabilidad permiten conocer la causa de esa desviación.

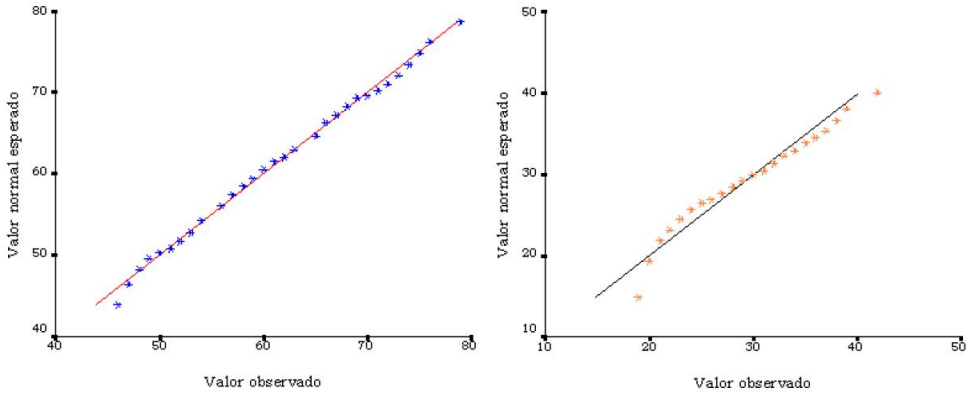


Figura B.3: Gráficos Q-Q.

Apéndice C

El Lenguaje de Programación Estadístico R

El *Lenguaje de Programación Estadístico R* (**R**) es un lenguaje de programación y un entorno para cálculos y gráficos estadísticos, similar al lenguaje **S-PLUS**. Proporciona muchas herramientas estadísticas (modelos lineales y no-lineales, pruebas estadísticas, algoritmos de clasificación, etc.) y produce gráficos de gran calidad. **R** es un ambiente de software libre, que se distribuye gratuitamente bajo los términos de *General Public Licence* (**GNU**). Su desarrollo y distribución son llevados a cabo por varios estadísticos conocidos como el *Grupo Nuclear de Desarrollo de R*. **R** compila y funciona en una amplia variedad de plataformas *Unix*, *Linux*, *Windows* y *MacOS*.

R, que es una implementación gratuita de **S-PLUS**, es un ambiente de programación que cuenta y permite desarrollar herramienta estadística, cuya base fue desarrollada en *Bell Labs.* (hoy *Lucent Technologies*) por R. Becker, J. Chambers y A. Wilks, en los años 70 y 80, que constituye el núcleo de **S-PLUS**, comercializado actualmente por la empresa *Insightful Corporation*. **R** carece de algunas de las facilidades de **S-PLUS**, pero tiene la enorme ventaja de ser gratuito y estar disponible en la red a través de *Comprehensive R Archive Network* (**CRAN**). Su dirección electronica es http://cran.r-project.org.

R juega un papel doble, como programa y lenguaje de programación. Para interactuar con el programa es necesario escribir expresiones que **R** interpreta y evalúa. **R** reconoce una gran variedad de expresiones, pero en uso interactivo las más comunes son nombres, que producen la definición de un objeto y llamadas a funciones, que llevan a cabo un cálculo especifico o ejecutan una serie de instrucciones.

R es un lenguaje *Orientado a Objetos*, interpretado (como Java) y no compilado (como C, C++, Fortran, etc.) lo cual significa que los comandos escritos en el teclado son ejecutados directamente sin necesidad de construir ejecutables.

R cuenta con librerías donde se tienen almacenadas y clasificadas una gran cantidad de funciones. Al momento de obtener **R** obtenemos también algunas librerías por defecto, las cuales se enlistan en el cuadro C.1. Con cada versión de **R** se agregan nuevas utilidades a cada librería o incluso una librería, en caso de que la versión con la que se cuenta no tiene alguna de las librerías se pueden descargar en la página de **CRAN**.

Cuadro C.1: Algunas librerías por defecto de **R**.

Librería	**Descripción**
abind	Combine multi-dimensional arrays
base	The R Base Package
boost	Boosting Methods for Real and Simulated Data
boot	Bootstrap R (S-Plus) Functions (Canty)
car	Companion to Applied Regression
graphics	The R Graphics Package
grid	The Grid Graphics Package
lattice	Lattice Graphics
lmtest	Testing Linear Regression Models
MASS	Main Package of Venables and Ripley's MASS

continuación...	
Librería	**Descripción**
nnet	Feed-forward Neural Networks and Multinomial Log-Linear Models
randomForest	Breiman and Cutler's random forests for classification and regression
RColorBrewer	ColorBrewer palettes
rgl 3D	visualization device system (OpenGL)
sandwich	Robust Covariance Matrix Estimators
splines	Regression Spline Functions and Classes
stats	The R Stats Package
survival	Survival analysis, including penalised likelihood
tcltk	Tcl/Tk Interface
tcltk2	SciViews GUI API - More tcltk
tkrplot	TK Rplot
tkWidgets	R based tk widgets
tools	Tools for Package Development
tree	Classification and regression trees
utils	The R Utils Package

Las funciones disponbibles están guardadas en una librería localizada en el directorio donde **R** está instalado. Este directorio contiene paquetes de funciones, las cuales a su vez están estructuradas en directorios. El paquete denominado **base** constituye el núcleo de **R** y contiene las funciones básicas del lenguaje para leer y manipular datos, algunas funciones gráficas y algunas funciones estadísticas. Cada paquete contiene un directorio denominado R con un archivo con el mismo nombre del paquete (por ejemplo, para el paquete **base**, existe el archivo C://Program Files/R/rw2001/library/base/R/base). Este archivo está en formato ***A****merican* ***S****tandard* ***C****ode for* ***I****nformation* ***I****nterchange* (**ASCII**) y contiene todas las funciones del paquete.

El cuadro C.2 enlista algunas funciones pertenecientes a la librería **base** en orden

alfabético con una pequeña descripción, a lo largo de esta sección se describiran más detalladamente algunas de las funciones.

Cuadro C.2: Algunas funciones de la librería **base**.

Función	**Descripción**
c	Combine Values into a Vector or List
cat	Concatenate and Print
data.frame	Data Frames
data.matrix	Data Frame to Numeric Matrix
date	System Date and Time
dir	List the Files in a Directory/Folder
else	Control Flow
for	Control Flow
force	Force evaluation of an Argument
function	Function Definition
identical	Test Objects for Exact Equality
if	Control Flow
ifelse	Conditional Element Selection
length	Length of an Object
matrix	Matrices
mean	Arithmetic Mean
max, min	Maxima and Minima
order	Ordering Permutation
paste	Concatenate Strings
print	Print Values
q	Terminate an R Session
rep	Replicate Elements of Vectors and Lists
rm	Remove Objects from a Specified Environment
sum	Sum of Vector Elements

continuación...	
Función	Descripción
vector	Vectors
write	Write Data to a File

C.1. Sintaxis

Como cualquier lenguaje de programación, **R** cuenta con sus propias reglas de sintaxis. El cuadro C.1 muestra las reglas para escribir operadores lógicos, donde x y y son vectores lógicos u objetos y las reglas para escribir operadores aritméticos donde x y y son números o vectores u objetos. Los operadores de relación se muestran en el cuadro C.1 donde x y y son vectores u otros objetos para los cuales se han escrito los métodos y por último estan los operadores de asignación, que se pueden consultar en el mismo cuadro, donde x representa el nombre de una variable y *value* el valor que se le desea asignar a x.

Cuadro C.3: Operadores lógicos y aritméticos.

Aritméticos	**Descripción**	**Lógicos**	**Descripción**
x + y	Suma.	! x	Negación.
x - y	Resta.	x & y	Operador and.
x * y	Multiplicación.	x && y	Operador and.
x / y	División.	x | y	Operador or.
x^y	Potencia.	x || y	Operador or.
x%% y	Modulo.	xor(x, y)	Operador del or exclusivo.
x%/% y	División del número entero.		

Si se desea utilizar **R** como ejecutor de comandos, esto es posible en modo consola, donde el cursor por defecto es el simbolo >, el cual indica que **R** está listo para recibir un comando. Si por el contrario se desea escribir un código más extenso, es

Cuadro C.4: Operadores de asignación y comparación.

Asignación	**Descripción**	**Comparación**	**Descripción**
x <− value	A la variable x se le asigna el valor que contenga *value*.	x < y	Menor que.
x <<− value	A la variable global x se le asigna el valor que contenga *value*.	x > y	Mayor que.
value −> x	A la variable x se le asigna el valor que contenga *value*.	x <= y	Menor o igual.
value −>> x	A la variable global x se le asigna el valor que contenga *value*.	x >= y	Mayor o igual.
x= value	A la variable x se le asigna el valor que contenga *value*.	x == y	Igualdad.
		x ! = y	Desigualdad.

recomendable escribirlo en un editor de textos, guardarlo de preferencia con extensión *.R* y darle a **R** la orden de ejecutar ese código por medio de la instrucción *source* que tiene como parametro de entrada la ruta del archivo que se desea ejecutar, por ejemplo
> source("C:/Tesis/ProgramaEjemplo1.R").

Un nombre es una combinación de letras, números y puntos, que no comienza con un número. **R** distingue mayúsculas de minúsculas, de modo que *a* y *A* son nombres distintos.
La llamada a una función se escribe usualmente como el nombre de la función seguido por un argumento que va colocado entre paréntesis. Describiremos algunas de las funciones proporcionadas por la librería **base**, ya que dicha librería cuenta con las funciones más básicas y comunes. Por ejemplo, la función *c* sirve para asignarle a una variable un conjunto de datos, en otras palabras crear un vector de datos, el parámetro de está función es un conjunto de elementos donde cada uno debe estar separado por medio de comas.

```
1> x = c(1, 2, 8, 3, 9, 16, 10)
```

La función *sum* tiene como parámetro un objeto de tipo vector numérico y regresa el resultado de la suma de los elementos del vector.

```
1> sum(x)
2 [1] 49
```

Dentro de la lista desplegada en el cuadro C.2 se presentan las funciones *max* y *min* donde ambas tienen como parámetro un vector tipo numérico y regresan el valor máximo y mínimo de los elementos del vector. La función *date* no tiene parámetro de entrada, sin embargo regresa la fecha del sistema. La función *mean* tiene como parámetro de entrada un vector numérico y regresa la media aritmética de los elementos del vector. El comando *sqrt* tiene como parámetro un vector numérico y

regresa el vector formado por las raices cuadradas de los elementos del vector de entrada. Para ordenar crecientemente o decrecientemente los elementos de un vector, existe la función *sort* donde sus parámetros de entrada son, el vector de elementos que se desea ordenar y la etiqueta indicadora del tipo de ordenación.

Para graficar existe la función *plot* que tiene varios parámetros de entrada, dentro del cuadro C.5 se enlistan, no es necesario indicar todos los parámetros, de echo si solo proporcionamos un solo vector númerico, como parámetro de entrada, la función nos despliega la gráfica correspondiente tomando el vector de entrada como las coordenadas en el eje y y el índice al cual corresponden dentro del vector como las coordenadas del eje x.

```
> max(x)
[1] 10
> min(x)
[1] 1
> date()
[1] "Thu Feb 16 23:19:40 2006"
> mean(x)
[1] 7
> sqrt(x)
[1] 1.000000 1.414214 2.828427 1.732051 3.000000 4.000000
[7] 3.162278
> sort(x, decreasing= TRUE)
[1] 16 10  9  8  3  2  1
> sort(x, decreasing= FALSE)
[1]  1  2  3  8  9 10 16
> plot(x)
```

A continuación veremos un ejemplo de una gráfica usando todos los parámetros

Cuadro C.5: Lista de parámetros de la función *plot*.

Parámetro	**Descripción**
x	Vector de coordenadas en x.
y	Vector de coordenadas en y.
type	Especifica el tipo de gráfico; "p": puntos, "l": línea, "b": puntos conectados por líneas, "o": igual al anterior pero las líneas están sobre los puntos, "h": líneas verticales, "s": escaleras, los datos se representan como la parte superior de las líneas verticales.
xlim	Especifica la cota inferior y superior del eje horizontal.
ylim	Especifica la cota inferior y superior del eje vertical.
xlab	Especifica un título para el eje horizontal.
ylab	Especifica un título para el eje vertical.
main	Especifica un título principal de la gráfica.
sub	Especifica un título secundario para la gráfica.
col	Indica el color para el tipo de gráfica indicado en *type*, por omisión es el color negro.
bg	Indica el color de fondo para la gráfica, por omisión es el color blanco.
pch	Controla el tipo de símbolo, ya sea un número entre 1 y 25 (los cuales traen asociados un símbolo), o un caracter.

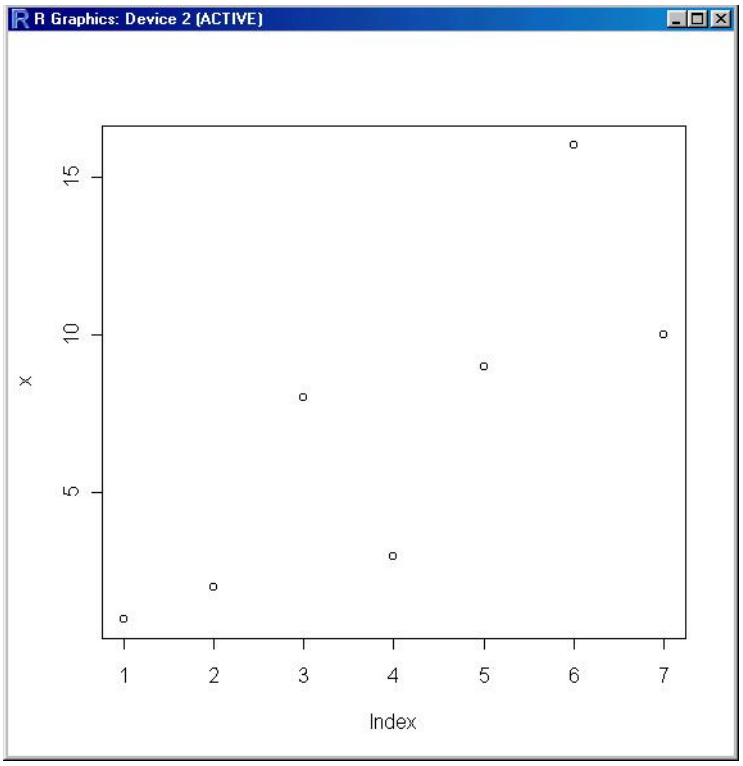


Figura C.1: Resultado del comando plot(x).

descritos en el cuadro C.5. La Figura C.2 muestra el resultado obtenido después de la ejecución del siguiente código.

```
> x= c(1,2,3,4,5,6,7)
> y= c(1,2,8,3,9,16,10)
> plot(x, y, type="p", xlim= c(0, 7), ylim= c(0, 17),
    xlab= "Coordenadas en x", ylab="Coordenadas en y",
    main= "Ejemplo de plot", sub= "Subtítulo de la gráfica",
    col= "blue", bg= "yellow", pch= "*")
```

Todas las expresiones de **R** producen un valor que se escribe automáticamente (aunque a veces es invisible). Muchas funciones, como *plot*(), *q*() se usan por sus efectos: graficar puntos, terminar una sesión de **R**, respectivamente. Si se escribe una expresión incompleta (por ejemplo, si se omite el segundo paréntesis al llamar una función), **R** presenta un símbolo de continuación: +, para indicar que falta algo

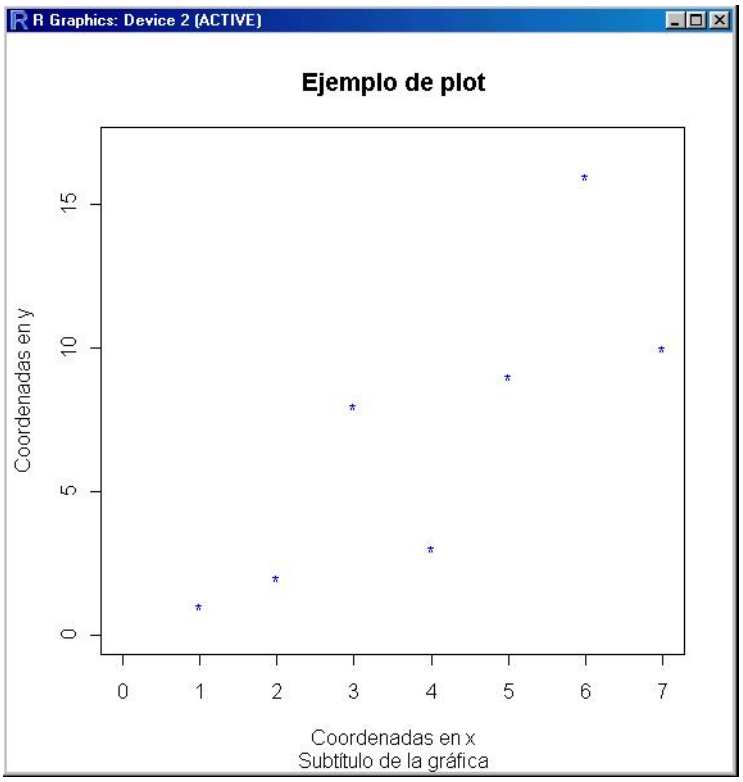


Figura C.2: Gráfica usando varios parámetros.

para completar la expresión. Por ejemplo

```
> sqrt(9
+
```

si completamos ahora la expresión el programa escribe el resultado.

```
> sqrt(9
+ )
[1] 3
```

Observamos que en los ejemplos anteriores las respuestas incluyen un índice, un número entero escrito entre corchetes. En cada línea de resultados de **R**, se incluye un índice que corresponde al primer resultado en esa línea. Así, cuando se escriben variables con muchos resultados, es más sencillo encontrar uno de ellos en particular.

R también cuenta con un caracter especial para indicar un comentario. Cualquier cosa que se escriba después del símbolo # en la línea de comandos es un comentario y el programa lo ignora.

Para ver el valor de una variable podemos escribir solamente el nombre de la variable o bien, usar la función *print*. Ejemplo

```
> a
[1] 10
> print(a)
[1] 10
```

Todas las asignaciones permanecen hasta que sean reemplazadas o eliminadas. El comando *rm* se usa para eliminar explícitamente una variable. Por ejemplo, *rm(a)* elimina la variable *a*. En el siguiente ejemplo cambiamos el valor de *a* y luego la eliminamos.

```
> a= 66
> print(a)
[1] 66
> rm(a)
> print(a)
Error: Object "a" not found
```

R guarda automáticamente las asignaciones de valores a variables, de modo que las variables pasan a ser objetos del programa y permanecen en el directorio de trabajo, a menos que sean eliminadas. Para saber cual es le directorio donde se esta trabajando, puede utilizar el comando *getwd*() (*get working directory*). Para cambiar el directorio de trabajo, se utiliza la función *setwd*() (*set working directory*); por ejemplo, setwd("C:/data") o setwd("/home/R"), es necesario proporcionar la

dirección completa del archivo si este no se encuentra en el directorio de trabajo. Al cerrar el programa aparece un mensaje preguntando si se desea guardar una *imagen del espacio de trabajo* (*workspace image*). Si se hace click en *Si* se guardan todos los objetos que están en el directorio de trabajo: los que estaban allí al inicio, más los que se hayan creado (y no se hayan eliminado) durante la sesión.

Una de las ventajas de **R** es la extensa documentación disponible en línea sobre las funciones y conjuntos de datos disponibles. Generalmente, el comando *help* muestra un conjunto de datos o un tópico sobre el cual se desea información. El comando *help()* abre una ventana que describe el uso de la función que aparezca como argumento, o las características del conjunto de datos, si es el caso. Por ejemplo *help*(*log*) abre una ventana que describe la sintaxis de las distintas funciones de **R** que calculan logaritmos, su estructura, la de las exponenciales (por ser las funciones inversas) y presenta referencias y ejemplos. Otras funciones que resultan útiles al buscar funciones que realicen determinadas tareas son *apropos()* y *help.search()*. Veamos ejemplos de su uso.

```
1> apropos("sort")
2 [1]"sortedXyData"  "is.unsorted"  "sort"  "sort.list"
```

El resultado que proporciona el comando *apropos* es una lista de las funciones cuyo nombre incluye los caracteres "sort". En cambio help.search("sort") abre una ventana donde aparecer una lista de todas aquellas funciones que tienen sort como alias o está palabra aparece en su título.

Orientado a objetos significa que las variables, datos, funciones, resultados, etc., se guardan en memoria en forma de objetos con un nombre específico. Y se pueden modificar o manipular estos objetos con operadores (aritméticos, lógicos y comparativos) y funciones (que a su vez también son objetos).

C.2. Algunas Funciones Importantes

En esta sección se describirán algunas funciones que son consideradas importantes y comunes.

C.2.1. La Función c

Esta función permite combinar varios elementos, del mismo tipo, para crear un vector. Basta listar los elementos en el orden deseado, separando cada elemento por una coma

```
> x= c(1, 1.5, 2, 2.5)
> print(x)
[1] 1.0 1.5 2.0 2.5
```

la función *c* también puede usarse con variables. Por ejemplo, si olvidamos incluir el número 3 como último elemento en el vector, podemos añadirlo de la siguiente manera

```
> x= c(x,3)
> print(x)
[1] 1.0 1.5 2.0 2.5 3.0
```

también puede usarse con caracteres

```
> y= c('esto', 'es', 'un', 'ejemplo')
> print(y)
[1] "esto" "es" "un" "ejemplo"
```

y con combinaciones de números y caracteres, aunque en este caso todos los elementos serán considerados como caracteres y en consecuencia no pueden realizarse operaciones aritméticas entre ellos.

```
> z= c(x, 'x')
> print(z)
[1] "1" "1.5" "2" "2.5" "3" x"
```

C.2.2. La Función seq

Esta función permite generar sucesiones crecientes o decrecientes de números reales. Su sintaxis es *seq*(cota inferior, cota superior, incremento). Ejemplos

```
> seq(0,100,5)
[1]  0  5 10 15 20 25 30 35 40 45 50 55 60
[14] 65 70 75 80 85 90 95 100
> seq(1955,1966,1)
[1] 1955 1956 1957 1958 1959 1960 1961 1962 1963 1964
[11] 1965 1966
> seq(10,12,0.2)
[1] 10.0 10.2 10.4 10.6 10.8 11.0 11.2 11.4 11.6 11.8
[11] 12.0
```

si no se especifica el incremento, el programa asume el valor uno por omisión. Si no se especifica un límite inferior, el programa también asume el valor uno.

```
> seq(1955,1966)
[1] 1955 1956 1957 1958 1959 1960 1961 1962 1963 1964 1965
[12] 1966
> seq(5)
[1] 1 2 3 4 5
```

C.2.3. La Función rep

Esta función sirve para repetir un patrón dado. Su sintaxis es *rep*(patrón, número de repeticiones). Veamos algunos ejemplos

```
> rep(10,3)
[1] 10 10 10
> rep(c(0,5), 4)
[1] 0 5 0 5 0 5 0 5
> rep(c("se", "va"),3)
[1] "se" "va" "se" "va" "se" "va"
> rep(1:5,2)
[1] 1 2 3 4 5 1 2 3 4 5
```

el número de repeticiones puede ser un vector, en cuyo caso debe tener el mismo número de componentes que el patrón y entonces cada elemento es repetido el número de veces correspondiente.

```
> rep(c(10,20),c(2,4))
[1] 10 10 20 20 20 20
> rep(1:3,1:3)
[1] 1 2 2 3 3 3
```

la función *rep* puede usarse para construir alguno de los vectores

```
> rep(1:3,rep(4,3))
[1] 1 1 1 1 2 2 2 2 3 3 3 3
```

si sabemos la longitud del vector que queremos obtener en lugar del número de veces que el patrón debe ser repetido, podemos usar la opción *length*, en lugar del número de repeticiones. Por ejemplo, si deseamos construir un vector de longitud diez usando el patrón 1 2 3 4 escribimos

```
> rep(c(1,2,3,4), length= 10)
 [1]  1  2  3  4  1  2  3  4  1  2
```

C.3. El Operador 'Índice'

Tomando en cuenta que para **R** todos los objetos son listas, es importante aclarar la ventaja que proporciona el operador índice. Sirve para extraer subconjuntos de objetos de datos en **R**. La sintaxis es *objeto[índice]*. En la expresión anterior, *objeto* puede ser cualquier objeto de datos en **R**, *índice* puede ser alguna de las tres siguientes alternativas, listadas a continuación, tomando en cuenta el ejemplo del conjunto de datos *letters*, que contiene las 26 letras minúsculas que se usan en el lenguaje inglés.

1. Enteros positivos, que corresponden a la posición que ocupan los datos buscados en el objeto. Ejemplo, para seleccionar la quinta letra escribimos

```
> letters[5]
 [1] "e"
> letters[c(2,8,16)]
 [1] "b" "h" "p"
```

2. Enteros negativos, que corresponden a la posición de los datos que deben ser excuidos. Ejemplo, para excluir la quinta letra escribimos

```
> letters [-5]
 [1] "a" "b" "c" "d" "f" "g" "h" "i" "j" "k" "l" "m" "n"
[14] "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
> letters [-c(2,8,16)]
 [1] "a" "c" "d" "e" "f" "g" "i" "j" "k" "l" "m" "n" "o"
```

```
[14] "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

3. Valores lógicos, valores verdaderos corresponden a los puntos que deseamos incluir en el subconjunto, valores falsos a los que deseamos excluir. Para presentar un ejemplo asignamos a la variable a los enteros del 1 al 26, por la asignación $a <- 1{:}26$. Luego verificamos que la expresión $a < 13$ produce una sucesión de 'T' y 'F' en la cual 'T' aparece en los primeros doce lugares y 'F' en el resto. Esto indica que los números en los doce primeros lugares satisfacen la desigualdad mientras que el resto no. Finalmente, usamos la expresión $a < 13$ como índice para obtener el subconjunto de las doce primeras letras.

```
> a= 1:26
> print(a)
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
[19] 19 20 21 22 23 24 25 26
> a < 13
 [1]  T  T  T  T  T  T  T  T  T  T  T  T  F  F  F  F  F  F  F  F  F  F  F  F  F  F
> letters [a < 13]
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l"
```

C.4. Operaciones aritméticas con vectores

R efectúa las operaciones aritméticas entre vectores componente a componente: si sumamos dos vectores de igual longitud el resultado es otro vector de la misma longitud, cuyas componentes son la suma de los componentes de los vectores que sumamos. De manera similar ocurre si realizamos cualquier otra operación aritmética, incluyendo la potenciación. Veamos algunos ejemplos

```
> a= 5:2
```

```
> b= (1:4)*2
> print(a)
[1] 5 4 3 2
> print(b)
[1] 2 4 6 8
> a + b
[1] 7 8 9 10
> a - b
[1] 3 0 -3 -6
> a * b
[1] 10 16 18 16
> a / b
[1] 2.50 1.00 0.50 0.25
> a ^ b
[1] 25 256 729 256
```

es posible realizar operaciones aritméticas entre un vector y un escalar, ejemplo

```
> 2 * a
[1] 10 8 6 4
```

también es posible realizar operaciones con más de dos vectores simultáneamente,

```
> d= rep(2,4)
> a + b + d
[1] 9 10 11 12
```

C.5. Tipos de datos

Toda expresión que escribimos en la consola es interpretada por **R** y produce un valor. Este valor es un objeto de datos ('data object') al cual puede asignársele un nombre. Los objetos de datos están compuestos por elementos, que en objetos simples corresponden a datos individuales y en objetos más complejos pueden consistir de otros objetos de datos. Los elementos pueden ser

- Lógicos, los valores T (cierto) o F (falso).
- Numéricos, números de punto flotante (con precisión doble). Los valores numéricos pueden escribirse como enteros (por ejemplo 4, -24), decimales (3.14, -2.717) o en notación científica (4e-23).
- Complejos, números complejos de la forma $a + bi$, donde a y b son numéricos (por ejemplo 3.6 + 2.4i).
- Caracteres, Sucesiones de caracteres limitados por comillas simples (' ') o dobles (" ") (por ejemplo 'letra', "prueba").

C.5.1. Matrices

Una matriz es un arreglo rectangular de celdas, cada una de las cuales contiene un valor, todos los valores dentro de la matriz son del mismo tipo. Para crear una matriz en **R** es posible usar la función *matrix*, cuya sintaxis es *matrix*(datos, nrow, ncol, byrow=F), donde *nrow* y *ncol* representan, respectivamente, el número de filas y columnas de la matriz. Sólo el primer argumento es indispensable. Si no aparecen ni el segundo, ni el tercero, los datos se colocan en una matriz unidimensional, es decir, en un vector. Si sólo uno de los valores se incluye, el otro se determina por división y se asigna, *datos* son los elementos que se guardarán en la matriz y por último *byrow*, un valor lógico, que indica si los *datos* serán acomodados siguiendo un orden por los renglones dentro de la matriz. Ejemplos

```
> matrix(1:6)
     [,1]
[1,]    1
[2,]    2
[3,]    3
[4,]    4
[5,]    5
[6,]    6
> matrix(1:6, nrow=3)
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

Veamos como se construyó esta última matriz. Los elementos de la matriz son los números del 1 al 6, listados en ese orden y queremos formar una matriz de tres filas. **R** hace la división del número de elementos entre el número de filas y obtiene que el número de columnas es dos. Para **R** los vectores son vectores columna y por esta razón los elementos de la lista se introducen primero en la primera columna y luego en la segunda. Es importante tener este orden en cuenta a la hora de hacer un programa. Si quisiéramos llenar la matriz por filas, es necesario poner 'T' como valor de *byrow*.

```
> matrix(1:6, nrow= 3, byrow= T)
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
```

los elementos de una matriz deben ser todos del mismo tipo. Para guardar columnas de distintos tipos en un arreglo bidimensional es necesario usar un cuadro de datos (data frame) que veremos más adelante.

Para sumar y restar matrices, operaciones que se realizan componente a componente, la única condición necesaria es que ambas matrices tengan las mismas dimensiones. Los símbolos usuales de suma y resta se usan para estas operaciones. Veamos un ejemplo

```
> A= matrix(1:6, nrow=3, byrow=T)
> B= matrix(seq(0, 10, 2), 3, 2)
> print(A)
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
> print(B)
     [,1] [,2]
[1,]    0    6
[2,]    2    8
[3,]    4   10
> A + B
     [,1] [,2]
[1,]    1    8
[2,]    5   12
[3,]    9   16
> A - B
     [,1] [,2]
[1,]    1    4
[2,]    1   -4
```

```
22 [3,]    1       -4
```

a pesar de la regla sobre la coincidencia de las dimensiones de las matrices que van a ser sumadas o restadas, es posible sumar o restar un escalar a una matriz

```
1> A + 2
2      [,1]    [,2]
3 [1,]    3       4
4 [2,]    5       6
5 [3,]    7       8
```

también es posible multiplicar o dividir por un escalar

```
1> B / 2
2      [,1]    [,2]
3  [1,]  0       3
4  [2,]  1       4
5  [3,]  2       5
```

la multiplicación de matrices es una operación más complicada y no se efectúa componente a componente. La sintaxis para la multiplicación de matrices es la siguiente

```
1> A %*%B
```

la regla para la multiplicación de matrices es que el número de columnas de la primera matriz debe ser igual al número de filas de la segunda. Por ejemplo, la matriz A que hemos definido anteriormente es una matriz de tres filas por dos columnas. La podemos multiplicar por cualquier matriz que tenga dos filas

```
1> D= matrix ((1:8)*3,2)
2> print(D)
```

```
     [,1] [,2] [,3] [,4]
[1,]    3    9   15   21
[2,]    6   12   18   24
> A %*% D
     [,1] [,2] [,3] [,4]
[1,]   15   33   51   69
[2,]   33   75   11  159
[3,]   51  117  183  249
```

C.5.2. Cuadros de Datos (Data Frames)

Los cuadros de datos permiten mezclar distintos tipos de datos en una misma estructura, de modo que se puede tener acceso a los datos como en una matriz. La sintaxis es *data.frame*(datos1, datos2, ...) los puntos suspensivos indican que pueden especificarse tantos conjuntos de datos como sea necesario. Para ver un ejemplo de la utilidad de los cuadros de datos veamos el ejemplo de unas compañías petroleras; observamos tres: Exxon Mobil, BP Amoco y Total ELF, las cuales son producto de fusiones recientes entre compañías que antes eran independientes. Queremos incluir esta información en nuestra estructura de datos añadiendo un vector con una nueva variable de caracteres. Pondremos F si la compañía es producto de una fusión y NF si no lo es. Inicialmente trataremos de incluir esta información en la matriz.

```
> Fusion= c("F", "NF", "F", "F", "NF")
> petro.datos1= cbind (petro.datos, Fusion)
> print(petro.datos1)
             Prod. Reservas Refin. Ventas Fusion
Exxon Mobil "4.3"   "21500"  "6.6"  "8.9"  "F"
Shell       "3.7"   "20500"  "3.4"  "5.7"  "NF"
BP Amoco    "4.1"   "19400"  "3.3"  "5.4"  "F"
Total ELF   "2.1"   "9600"   "2.4"  "3.2"  "F"
```

```
Chevron      "1.5"    "6200"     "1.6"    "2.0"    "NF"
```

el problema con este procedimiento es que, debido a la estructura de manejo de distintos modos de datos, los datos han sido transformados a modo 'carácter' y no podemos hacer operaciones aritméticas con ellos.

C.5.3. Listas

Las listas son objetos que permiten unir información que no tiene la misma estructura. Recordemos que en los cuadros de datos, los datos podían ser de distintos tipos pero debían tener la misma estructura. Regresemos al ejemplo de las compañías petroleras y supongamos que queremos incluir la lista de las tres compañías estatales latinoamericanas más importantes: PDVSA, Pemex y Petrobras. Esta información no tiene la misma estructura que la anterior, pero usando una lista podemos almacenarlas conjuntamente.

```
> Com.Lam= c("PDVSA", "Pemex", "Petrobras")
> petro.list= list(petro.datos1, Com.Lam)
> print(petro.list)
[[1]]:
             Prod.   Reservas    Refin.    Ventas    Fusion
Exxon Mobil "4.3"     "21500"   "6.6"     "8.9"       "F"
Shell       "3.7"     "20500"   "3.4"     "5.7"       "NF"
BP Amoco    "4.1"     "19400"   "3.3"     "5.4"       "F"
Total ELF   "2.1"     "9600"    "2.4"     "3.2"       "F"
Chevron     "1.5      "6200"    "1.6"     "2.0"       "NF"
[[2]]:
[1] "PDVSA" "Pemex" "Petrobras"
```

buscamos ahora algunos valores en la lista

```
> petro.list$Com.Lam
NULL
> petro.list[[1]]["Chevron", "Refin"]
[1] 1.6
> petro.list[[1]]["Chevron", "Fusion"]
[1] NF Levels: [1] "F" "NF
> petro.list[[2]]
[1] "PDVSA" "Pemex" "Petrobras"
```

el comando lista simplemente 'pega' las distintas piezas incluidas en el comando. Crea un índice para cada pieza, que se listan secuencialmente y pueden ser recuperados usando corchetes dobles. Observe que la sintaxis usada con los cuadros de datos no funciona con listas y el programa devuelve el valor NULL. Para tener acceso a datos individuales tenga en cuenta que los corchetes dobles deben ser tratados como parte del nombre de la estructura de interés. El comando petro.list[[2]] produce el segundo elemento de la lista. De allá en adelante las referencias son como si se tratara de una matriz

```
> petro.list[[1]][3,2]
[1] 19400
```

nos da las reservas de BP Amoco.

C.6. Programando con R

Hasta este momento ya contamos con las ideas generales del funcionamiento de **R**, usaremos está información para utilizar **R** como lenguaje de programación. Como ya se menciono, es recomendable escribir el código en un editor de texto (el compilador de **R** para *Windows* trae uno por omisión) y guardarlo con extensión ".R".

Como en otros lenguajes, **R** posee estructuras de control que no son muy diferentes a las de un lenguaje de bajo nivel como **C**. Supongamos que tenemos un vector **X** y para cada elemento de **X** con valor igual a b queremos asignar el valor cero o uno en caso contrario, a otra variable **Y**.

```
Y <- numeric(length(X))
for(i in 1:length(X))
  if(X[i] == b)
    Y[i] <- 0
  else
    Y[i] <- 1
```

también se podría escribir como

```
Y <- numeric(length(X))
for(i in 1:length(X))
  Y[i] <- ifelse(X[i] == b, 0, 1)
```

Se pueden usar llaves para ejecutar varias instrucciones

```
for(i in 1:length(X))
{
  Y[i] <- 0
  ...
}
if(X[i] == b)
{
    Y[i] <- 0
    ...
}
```

Otra opción es ejecutar una instrucción siempre y cuando se cumpla una cierta condición

```
while(a > b)
{
    ...
}
```

Sin embargo, este tipo de ciclos y estructuras se pueden evitar gracias a una caracteristica clave en **R**, la cual es donde radica su gran ventaja sobre otros lenguajes de programación: la vectorización. La vectorización hace los ciclos implicitos en las expresiones. Por ejemplo, consideremos la suma de dos vectores.

```
X= c(1, 5, 9)
Y= c(3, 5, 7)
Z= X + Y
```

esta suma también se puede realizar con ciclos como en cualquier otro lenguaje

```
X= c(1, 5, 9)
Y= c(3, 5, 7)
Z= numeric(length(X))

for(i in 1: length(Z))
  Z[i]= X[i] + Y[i]
```

en este caso es necesario crear con anterioridad el vector **Z** por la necesidad de indexar los elementos. En ambos casos es necesario que los vectores **X** y **Y** sean del mismo tamaño ya que de no serlos ocurriria un error, indicando que los vectores no tiene la misma longitud.

Las ejecuciones condicionales *if* ... *else*, del primer ejemplo, se pueden evitar con el uso de indexación lógica

```
Y[X == b]=   0
Y[X != b]=   1
```

Para crear una función es necesario indicar primero que esto se va a realizar, por medio del comando *function* donde los parámetros que le pongamos a esté comando serán los parámetros de entrada a la función que definamos. Ejemplo

```
SumaDosVectores <- function(x, y)
{
  Z= numeric(length(x))
  for(i in 1: length(Z))
    Z[i]= x[i] + y[i]
  return(Z)
}
X= c(1, 5, 9)
Y= c(3, 5, 7)

Resultado= SumaDosVectores(X, Y)
```

supongamos que el código anterior se ha guardado con el nombre "PrimerEjemplo.R"para ejecutarlo, hay que cargarlo por medio del comando *source*.

```
> source("PrimerEjemplo.R")
> print(Resultado)
[1]  4 10 16
```

lo anterior funciona si el programa esta guardado en el mismo directorio de trabajo que estamos trabajando de no ser así, es necesario escribir toda la ruta del archivo.

Existen dos formas de especificar argumentos a una función: por sus posiciones o por sus nombres. Por ejemplo, consideremos una función como tres argumentos

```
Fun <- function(arg1, arg2, arg3)
{
  ...
}
```

Fun se puede ejecutar sin usar los nombres arg1, arg2 y arg3, si los objetos correspondientes están colocados en la posición correcta; por ejemplo: Fun(x, y, z). Sin embargo, la posición no tiene ninguna importancia si se utilizan los nombres de los argumentos explicitamente; por ejemplo Fun(arg3= z, arg2= y, arg1= x). Otro rasgo importante de las funciones es la posibilidad de usar valores por defecto en la definición; por ejemplo

```
Fun <- function(arg1, arg2= 16, arg3= FALSE)
{
  ...
}
```

Fun(x) y Fun(x, 16, FALSE) producirán el mismo resultado. Por ejemplo

```
Fun2 <- function(a, veces= 5)
{
  for(i in 1: veces)
  {
    Sig= (1/ (1 + exp(-i))) + a
    print(Sig)
  }
}
```

suponiendo que Fun2 es una función que se agregó al archivo "PrimerEjemplo.R", la función Fun2 podemos llamarla de dos formas

```
> source("PrimerEjemplo.R")
> Fun2(1)
[1] 1.731059
[1] 1.880797
[1] 1.952574
[1] 1.982014
[1] 1.993307
> Fun2(2, 3)
[1] 2.731059
[1] 2.880797
[1] 2.952574
```

en la primera llamada el parámetro *veces* toma el valor por defecto y en la segunda llamada toma el valor correspondiente a su posición.

C.7. Paquete Tcl/Tk

Para realizar la parte gráfica del programa se utilizó el paquete *Tcl/Tk* para **R**, con el cual se pueden generar ventanas y elementos visuales, todo lo que revisamos en las secciones anteriores solo dan resultados en modo consola. Dentro de esta sección hablaremos un poco del uso de esta paquetería.

Para poder hacer uso de *Tcl/Tk* primero hay que cargarla a **R**, por medio de la instrucción library(tcltk), si no se cuenta con el paquete se puede obtener de la página oficial de **CRAN**. El cuadro C.6 enlista algunas de las funciones con las que cuenta está librería.

Cuadro C.6: Algunas funciones de la librería *TclTk*.

Función	Descripción
tclvalue	Low-level Tcl/Tk Interface
tclVar	Low-level Tcl/Tk Interface
tkadd	Tk widget commands
tkaddtag	Tk widget commands
tkbind	Tk non-widget commands
tkbutton	Tk widgets
tkcanvas	Tk widgets
tkcheckbutton	Tk widgets
tkconfigure	Tk widget commands
tkcoords	Tk widget commands
tkcreate	Tk widget commands
tkcurselection	Tk widget commands
tkdelete	Tk widget commands
tkdeselect	Tk widget commands
tkdestroy	Low-level Tcl/Tk Interface
tkfocus	Tk non-widget commands
tkframe	Tk widgets
tkget	Tk widget commands
tkgetOpenFile	Tk non-widget commands
tkgetSaveFile	Tk non-widget commands
tkindex	Tk widget commands
tkinsert	Tk widget commands
tklabel	Tk widgets
tklistbox	Tk widgets
tkmenu	Tk widgets
tkmenubutton	Tk widgets
tkmessage	Tk widgets

continuación...	
Función	Descripción
tkmessageBox	Tk non-widget commands
tkmove	Tk widget commands
tkpack	Tk non-widget commands
tkpager	Page file using Tk text widget
tkpopup	Tk non-widget commands
tkpost	Tk widget commands
tkradiobutton	Tk widgets
tkscrollbar	Tk widgets
tksearch	Tk widget commands
tkselect	Tk widget commands
tksize	Tk widget commands
tktext	Tk widgets
tktitle	Tk non-widget commands
tktoplevel	Tk widgets
tktype	Tk widget commands

Antes que todo, el primer paso es crear una ventana, en la cual vamos a estar trabajando, esto se hace con el comando *tktoplevel*, su sintaxis es tktoplevel(width, height, bg), el cuadro C.7 muestra los parámetros, que regresa un objeto tipo *tkwin*, propio de la libreriía *TclTk* que se refiere a una ventana, para poder ponerle un título a está nueva ventana existe el comando *tktitle*, que recibe como parámetro el nombre del objeto al cual se le va asignar un título. Veamos el siguiente ejemplo

```
1 Ventana= tktoplevel(width= 600, height= 400, bg= "yellow")
2 tktitle(Ventana)= "CSR"
```

la ejecución de estos dos comandos dan como resultado la Figura C.3. Como podemos ver es solo un recuadro vació de color amarillo, el cual se especifico con el uso

Cuadro C.7: Parámetros para *tktoplevel*.

Parámetro	Descripción
width	Indica el ancho de la ventana.
height	Indica lo alto de la ventana.
bg	Indica el color de fondo que tendrá de la ventana.

del parámetro *bg*, al cual a continuación le vamos agregar un botón. Para agregarle

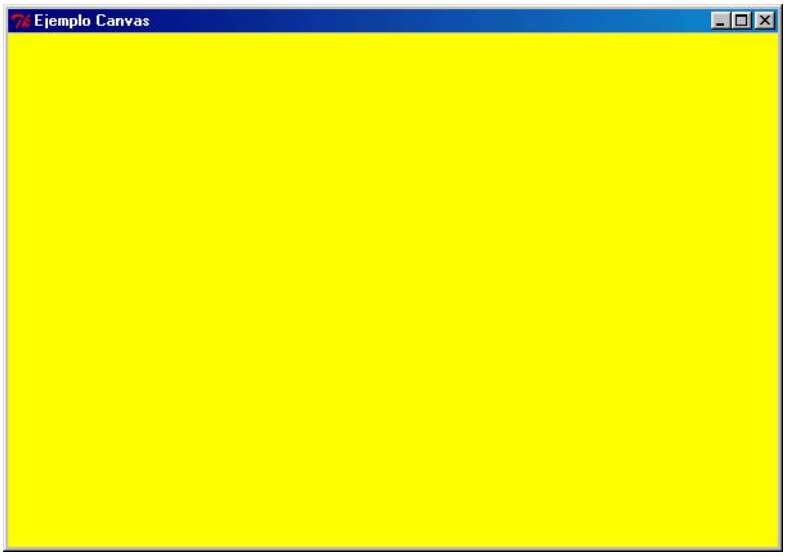


Figura C.3: Nueva Ventana.

un botón a la ventana anterior, hay que hacer uso de la función *tkbutton*, su sintaxis es *tkbutton*(parent, text, width, command), el cuadro C.8 muestra la descripción de los parámetros de entrada a la función. Ejemplo

```
1 Close.but= tkbutton(Ventana, text= "Cerrar", width= 10,
2 command= function() tkdestroy(Ventana))
3 tkgrid(Close.but)
```

Close.but es nombre de variable, el punto no significa que estemos accesando algún atributo, en este caso la acción que va realizar al ser precionado es el de de-

Figura C.4: Botón.

Cuadro C.8: Parámetros para *tkbutton*.

Parámetro	Descripción
parent	Nombre del objeto donde se va agregar el botón.
text	Texto que aparecerá dentro del botón.
width	Indica el ancho del botón.
command	Acción que va a realizar al momento de ser precionado.

struir la ventana, esto se indica por medio de la función *tkdestroy* que tiene como parámetro el nombre del objeto que va destruir. La función *tkgrid* se encarga de agregar el nuevo componente a la ventana actual.

Con la función *tkcanvas* se proporciona un lienzo para realizar dibujos, como por ejemplo: cuadros, ovalos, rectangulos y líneas. La sintaxis es *tkcanvas*(parent, width, height, bg), el cuadro C.9 muestra la descripción de los parámetros de entrada a la función. Ejemplo

```
1 TopCanvas= tktoplevel()
2 tktitle(TopCanvas)= "Ejemplo Canvas"
3 canvas= tkcanvas(TopCanvas, width=500, height=300, bg="red")
4 tkgrid(canvas)
```

la Figura C.5 muestra el resultado obtenido después de la ejecución del código anterior. Se puede observar que el espacio de color rojo, es el correspondiente al lienzo que se agrego a la ventana. Para agregar componentes al lienzo es necesario utilizar

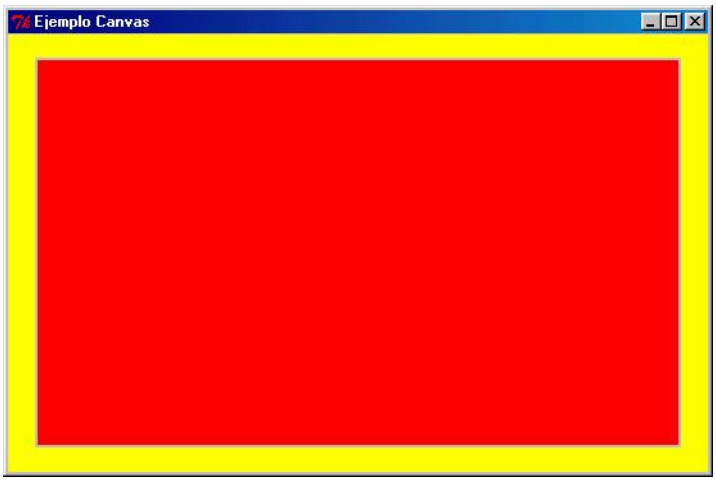


Figura C.5: Ejemplo de Canvas.

Cuadro C.9: Parámetros para *tkcanvas*.

Parámetro	Descripción
parent	Nombre del objeto donde se va agregar el lienzo.
width	Indica el ancho del lienzo.
height	Indica lo alto del lienzo.
bg	Indica el color de fondo que tendrá el lienzo.

la función *tkcreate* con la cual creamos un componente y lo insertamos dentro del área de dibujo, la sintaxis de esta función depende del tipo de componente que se quiere agregar, el cuadro C.10 muestra los parámetros de entrada dependiendo del tipo de componente que se desee agregar al lienzo. Ejemplo, la Figura C.6 muestra el resultado obtenido de la implementación del siguiente código.

```
X= c(100, 100, 300, 200)
Y= c(100, 150, 150, 250)
Oval= tkcreate(canvas,"oval",X[1]-6,Y[1]-6,X[1]+6,Y[1]+6,
  fill= "yellow")
tkaddtag(canvas, "O", "withtag", Oval)
Line= tkcreate(canvas,"line",X[2],Y[2],X[3],Y[3], width=2,
  fill= "pink")
tkaddtag(canvas, "L", "withtag", Line)
Text=tkcreate(canvas,"text",X[4],Y[4], text="Ejemplo canvas!",
  anchor= "nw")
tkaddtag(canvas, "T", "withtag", Text)
```

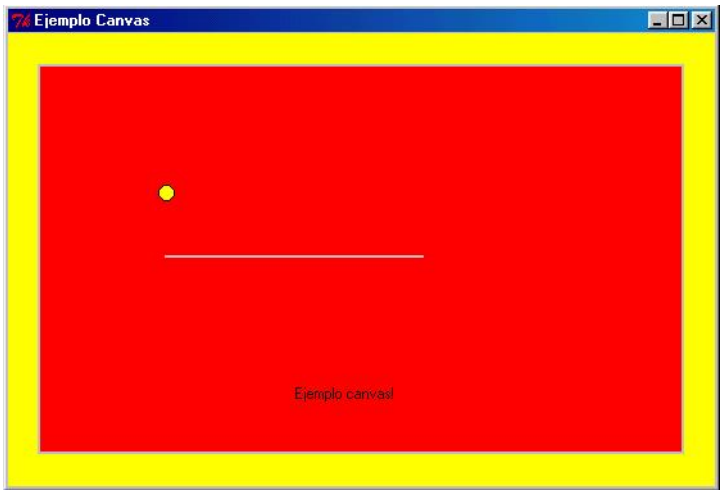


Figura C.6: Canvas con componentes.

la función *tkaddtag* es la encargada de agregar al lienzo el componente creado con *tkcreate*, su sintaxis es tkaddtag(parent, objeto, tipo, componente), el cuadro C.11

muestra la descripción de los parámetros para esta función.

Cuadro C.10: Parámetros para *tkcreate*.

comunes	
Parámetro	Descripción
parent	Nombre del objeto donde se va agregar el componente.
type	Indica el tipo de componente se desea agregar.
dependen del tipo	
type= oval	x_1, y_1; Coordenadas del primer punto. x_2, y_2; Coordenadas del segundo punto. fill; Color con el cual se va a rellenar el ovalo.
type= line	x_1, y_1; Coordenadas del punto de inicio. x_2, y_2; Coordenadas del punto final. fill; Color con el cual se va a dibujar la línea. width; Ancho de la línea.
type= text	x, y; Coordenadas de donde va a iniciar el texto. text; Texto que se quiere agregar. anchor; Indicador de hacia donde va a estar orientado el texto.

A continuación veremos un código un poco más elaborado, en el cual pondremos dos nodos (ovalos) unidos por medio de una línea bicolor, estos nodos se podrán manipular por medio del uso del ratón, puesto que la línea esta unida a los extremos de cada nodo, está se estirará conforme se modifique la posición de uno de los nodos.

Ejemplo completo

```
1 #manda llamar a la librería
2 library(tcltk)
3 TopCanvas=tktoplevel(width=600, height=400, bg="yellow")
```

Cuadro C.11: Parámetros para *tkaddtag*.

Parámetro	Descripción
parent	Nombre del objeto donde se va agregar el componente.
objeto	Objeto tipo etiqueta.
tipo	Indica que esta relacionado con el objeto.
componente	Nombre del componente que se va a gregar.

```
tktitle(TopCanvas)="Ejemplo Canvas"
canvas=tkcanvas(TopCanvas,width=500,height=300,bg="red")
tkgrid(canvas)
from=c(1,2);to=c(2,3)
color=c("pink","black")
X=c(100,225,350);Y=c(100,150,200)
nodeEdges=vector("list",4)
nodeItem=Lin=vector("character",2)

Mueve<<-function(x,y,Nodo)
{
   for(e in nodeEdges[[Nodo]] )
      tkcoords(canvas,e$edgeItem,x,y,
        X[e$to],Y[e$to])
   tkmove(canvas,nodeItem[Nodo],x-X[Nodo],y-Y[Nodo])
   X[Nodo]<<-x
   Y[Nodo]<<-y
}

moveNode<<-function(i)
{
```

```
25    force(i)
26    function(x,y)
27    {
28       x=as.numeric(x);y=as.numeric(y)
29       Mueve(x,y,i)
30       xmedio<<-((X[1])+(X[3]))/2
31       ymedio<<-((Y[1])+(Y[3]))/2
32       Mueve(x=xmedio,y=ymedio,Nodo=2)
33    }
34 }
35
36 LineaColor<<-function(i)
37 {
38    f=from[i];t=to[i]
39    r=tkcreate(canvas,"line",X[i],Y[i],X[i+1],Y[i+1],
40       width=2,fill=color[i])
41    tag=paste("linea",i,sep="")
42    tkaddtag(canvas,tag,"withtag",r)
43    Lin[i]<<-tag
44    nodeEdges[[f]]<<<-c(nodeEdges[[f]],
45       list(list(to=t,edgeItem=Lin[i])))
46    nodeEdges[[t]]<<<-c(nodeEdges[[t]],
47       list(list(to=f,edgeItem=Lin[i])))
48 }
49
50 LineaColor(1);LineaColor(2)
51
52 for(i in c(1,3))
53 {
54    p=tkcreate(canvas,"oval",X[i]-6,Y[i]-6,X[i]+6,
```

```
55      Y[i]+6, fill="pink")
56    l=tkcreate(canvas,"text",X[i]+6,Y[i],text=
57      paste("nodo",i,sep=""),anchor="nw")
58    tag= paste("node",i,sep="")
59    tkaddtag(canvas,tag,"withtag",p)
60    tkaddtag(canvas,tag,"withtag",l)
61    nodeItem[i]=tag
62    tkitembind(canvas,p,"<B1-Motion>",moveNode(i))
63  }
```

A continuación se realiza una pequeña descripción del código, la Figura C.7 muestra el resultado obtenido.

Linea 1: Manda llamar la paquetería *tcltk*.

Linea 2 a 5: Se crea la ventana y se le agrega un lienzo, con las mismas dimenciones que en los ejemplos anteriores.

Linea 6: Se establecen los extremos de las líneas.

Linea 7: Colores que se van a usar.

Linea 8: Coordenadas de los nodos.

Linea 9 y 10: Se especifican las variables.

Linea 12 a 19: Cuerpo de la función Mueve.

Linea 14 y 15: Ciclo en el cual se obtienen las coordenadas de los nodos.

Linea 16: Mueve el nodo hacia las nuevas coordenadas establecidas.

Linea 17 y 18: Ajusta los valores de las coordenadas.

Linea 21 a 32: Cuerpo de la función MoveNode.

Linea 23: Fuerza la evaluación del argumento formal.

Linea 24 a 31: Cuerpo de la función interna de MoveNode.

Linea 26: Captura los valores de las coordenadas (x, y), del nodo en cuestión.

Linea 27: Llamado a la función Mueve.

Linea 28 y 29: Recalcula el punto medio entre los nodos extremos.

Linea 30: Llamado a la función Mueve.

Linea 34 a 43: Cuerpo de la función interna de LineaColor.

Linea 36: Captura el punto inicial y el final, que forman la línea.

Linea 37: Crea un componente tipo línea.

Linea 38: Establece un nombre al componente.

Linea 39: Agrega el componente al lienzo.

Linea 40: Agrega el nombre del componente a una lista.

Linea 41 y 42: Agrega a la lista de nodos.

Linea 45: Llamados a la función LineaColor, para agregar las dos líneas.

Linea 47 a 56: Ciclo en el cual se crean los nodos y se les asigna un comportamiento.

Linea 49: Crea el componente tipo nodo.

Linea 50: Crea el componente tipo texto.

Linea 51: Se crea la etiqueta a la cual se le agregarán los componentes.

Linea 52 y 53: A la etiqueta se le agregán los dos componentes creados en las líneas 49 y 50.

Linea 54: La etiqueta se almacena en un arreglo de etiquetas.

Linea 55: Se establece la acción que se va efectuar sobre el nodo al ser precionado con el ratón.

La linea 55 introduce un nuevo comando *tkitembind*, su sintaxis es tkitembind(parent, componente, evento, acción), la descripción de sus parámetros se pueden ver en el cuadro C.12.

Cuadro C.12: Parámetros para *tkitembind*.

Parámetro	Descripción
parent	Nombre del objeto donde se va agregar el componente.
componente	Nombre del componente al cual se le va aplicar una acción.
evento	Bajo que evento del ratón se va a generar la acción.
acción	Operación que se realizará al cumplirse el evento implicado con el componente.

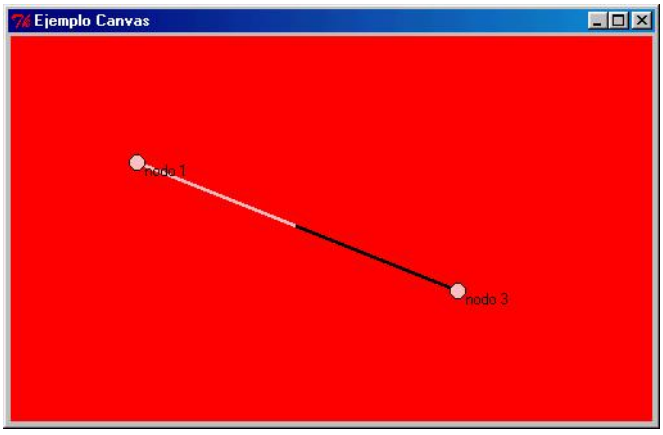


Figura C.7: Resultado arrojado por el código.

Bibliografía

[1] **Varios Autores**, *Artificial Neural Networks*, North Holland, 1991.

[2] **David M. Skapura** *Building Neural Networks*, Addison Wesley, 1996

[3] **Breiman, Leo**, *Classification and Regression Trees*, Chapman, 1998.

[4] **Johan J. L. Van Horebeek y Hugh Chipman**, *Context Sensitive Regression Models with Binary Predictors*. Disponible como archivo pdf en http://ace.acadiau.ca/math/chipmanh/papers/context-sensitive-regression-2005.pdf. Consultado por última vez en Enero de 2006.

[5] **Terrence L. Fine**, *Feedforward Neural Network Methodology*, Springer Verlag, 1999.

[6] **Marko Robnik-Sikonja**, *Improving Random Forests*, ECML 2004, LNAI 3210, Springer, Berlin, 2004, pp. 359-370. Disponible como archivo pdf en http://lkm.fri.uni-lj.si/rmarko/papers/robnik04-ecml.pdf. Consultado por última vez en Enero de 2006.

[7] **Alexander McFarlane Mood**, *Introducción a la Teoría de la Estadística*, Aguilar, 1965.

[8] **Douglas C. Montegomery & Elizabeth A. Peck**, *Introduction to Linear Regression Analysis*, John Wiley, 1982.

[9] **The R core development team**, *Lenguaje de Programación Estadístico R*. Pagina oficial www.r-project.org

[10] **B. D. Ripley**, *Pattern Recognition and Neural Networks*, Cambridge, University Press, 1996.

[11] **Leo Breiman y Adele Cutler**, *Random Forests*, Machine Learning, 45, 5-32, 2001. Pagina oficial http://stat-www.berkeley.edu/users/breiman/RandomForests/. Consultado por última vez en Enero de 2006.

[12] **Leo Breiman**, *Statistical Modeling: The Two Cultures*, Statistical Science, 2001, Vol. 16, No 3, 199-231. Disponible como archivo pdf en www.cis.upenn.edu/group/datamining/ReadingGroup/papers/breiman2001.pdf. Consultado por última vez en Enero de 2006.