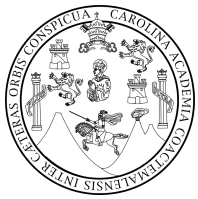


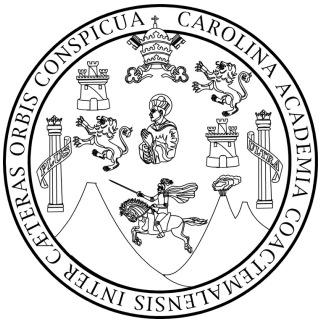
Universidad de San Carlos de Guatemala
Escuela de Ciencias Físicas y Matemáticas
Departamento de Matemática

MÉTODOS SELECTOS PARA CATEGORIZAR CRÉDITO

José Alberto Ligorría Taracena
Asesorado por Lic. José Carlos Alberto Bonilla Aldana

Guatemala, noviembre de 2016

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA



ESCUELA DE CIENCIAS FÍSICAS Y MATEMÁTICAS

MÉTODOS SELECTOS PARA CATEGORIZAR CRÉDITO

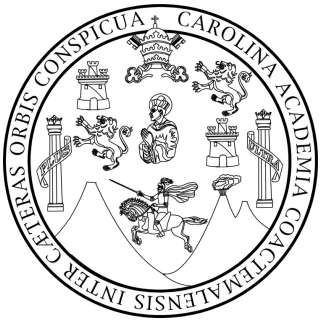
TRABAJO DE GRADUACIÓN
PRESENTADO A LA JEFATURA DEL
DEPARTAMENTO DE MATEMÁTICA
POR

JOSÉ ALBERTO LIGORRÍA TARACENA
ASESORADO POR LIC. JOSÉ CARLOS ALBERTO BONILLA ALDANA

AL CONFERÍRSELE EL TÍTULO DE
LICENCIADO EN MATEMÁTICA APLICADA

GUATEMALA, NOVIEMBRE DE 2016

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
ESCUELA DE CIENCIAS FÍSICAS Y MATEMÁTICAS



CONSEJO DIRECTIVO

DIRECTOR	M.Sc. Edgar Anibal Cifuentes Anléu
SECRETARIO ACADÉMICO	Ing. José Rodolfo Samayoa Dardón

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

EXAMINADOR	Lic. William Roberto Gutiérrez Herrera
EXAMINADOR	Lic. Hugo Allan García Monterrosa
EXAMINADOR	Lic. Alan Gerardo Reyes Figueroa

AGRADECIMIENTOS A:

Dios	Por darme la vida y la oportunidad de aprender matemática.
Mis padres	Oscar Ligorría y Dorys Taracena de Ligorría, por el amor que me dedicaron, por apoyarme en todo momento y por confiar en mí.
Mis catedráticos	Y en particular a mi asesor, el Lic. José Bonilla, por compartir sin recelo sus conocimientos y experiencias académicas.
Mi novia	Flor Pérez, por su apoyo durante las fases de mi preparación universitaria y compartir conmigo los buenos y malos momentos.
Mis familiares y amigos	Por compartir conmigo tantos momentos y proyectos dentro o fuera de las aulas universitarias.
Universidad de San Carlos de Guatemala	Por darme la oportunidad de ser parte de esta casa de estudios.
Victor Hugo	Por darle la oportunidad de ser parte de su equipo y aprender lo necesario para la realización de este trabajo.

TRABAJO QUE DEDICO A:

Mi familia Tanto a aquellos que pertenecen a ella actualmente, como a quienes algún día serán parte de ella. Son la base de mi vida.

Universidad de San Carlos de Guatemala
Escuela de Ciencias Físicas y Matemáticas

Ref. D.DTG. 010-2016
Guatemala 18 de noviembre de 2016

El Director de la Escuela de Ciencias Físicas y Matemáticas de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Coordinador de la Licenciatura en Matemática Aplicada, autoriza la impresión del trabajo de graduación Titulado: **MÉTODOS SELECTOS PARA CATEGORIZAR CRÉDITO** presentado por el estudiante universitario: **José Alberto Ligorría Taracena.**

IMPRIMASE.

UNIVERSIDAD DE SAN CARLOS ESCUELA DE CIENCIAS FISICAS Y MATEMÁTICAS DIRECCIÓN DE GUATEMALA

MsC. Edgar Aníbal Cifuentes Anleu
Director
Escuela de Ciencias Físicas y Matemáticas

/pec

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ALGORITMOS

LISTA DE SÍMBOLOS

Símbolo	Significado
$:=$	es definido por
$=$	es igual a
$\approx$	es aproximadamente igual a
$\Rightarrow$	implicación
$\Leftrightarrow$	si, y sólo si,
$\mathbb{Z}$	conjunto de los números enteros
$\mathbb{Z}^+$	conjunto de los números enteros positivos
$\mathbb{Q}$	conjunto de los números racionales
$\mathbb{R}$	conjunto de los números reales
$\mathbb{N}$	conjunto de los números naturales
$\langle a, b\rangle$ o $a \cdot b$	producto interno de los vectores a, b en un espacio de Hilbert
$\sum_C A$	suma los términos A que cumplan las condiciones C. Si los términos son a_i y las condiciones $1 \leq i \leq k, i \in \mathbb{Z}$ escribiremos $\sum_{i=1}^{k} a_i$
$\geq$	mayor o igual que
$\leq$	menor o igual que
$\lvert a\rvert$	valor absoluto del número a
$\lVert A\rVert$	norma del vector A
$\lVert x\rVert_2$	norma euclidiana del vector x, si $x = \{x_1, x_2, ..., x_n\}$ entonces $\lVert x\rVert_2 = \sqrt{\sum_{i=1}^{n}(x_i)^2}$
A^T	transpuesta los elementos de la matriz A
A^{-1}	matriz inversa de la matriz A
$\subset$	contenido en
$\supset$	contiene a
$\subseteq$	contenido o igual
$\supseteq$	contiene o igual a
$\varnothing$	conjunto vacío

Símbolo	Significado
A^c	complemento de A
$\mathbb{E}$	conjunto de clases que pueden tomar las variables categóricas
X_n	vector de observaciones n de un conjunto de observaciones
x_j	variable j de la observación X
x_j^k	valor de la variable j para la observación X_k
Y_n	observación categórica de objetivo
t	nodo
t_0	nodo raíz
t_p	nodo padre
t_i	nodo hijo del lado izquierdo
t_d	nodo hijo del lado derecho
tt_c	conjunto de nodos hijos por una partición sobre variable categórica
P_i	proporción de datos del hijo izquierdo respecto al padre
P_d	proporción de datos del hijo derecho respecto al padre
i	impureza
i_G	impureza definida como el ínidice Gini
i_E	impureza definida con la entropía
i_e	impureza definida con el error de clasificación
Δi	diferencia de impurezas
T_0	primer árbol creado
T	sub-árbol del árbol T_0
$\overline{T}$	conjunto de nodos hoja del árbol T
$\|T\|$	cardinalidad del conjunto T, si T es un árbol es la cantidad de nodos que tiene
T_t	sub-árbol del árbol T_0, cuyo nodo raíz es t
$\setminus$	diferencia de conjuntos (árboles)
$-$	cuando se usa entre un árbol y un sub-árbol representa la poda
N_m	cantidad de observaciones en el nodo final m
N	cantidad de observaciones en el conjunto de entrenamiento
R_{t_m}	estimador del error local de clasificación para un nodo terminal t_m de T
$R(T)$	estimador del error global de clasificación de T
α	parámetro de complejidad
α_k	paremétro de complejidad usado como límite

Símbolo	Significado
$C_\alpha(T)$	medida de costo complejidad de un sub-árbol T
$C_\alpha(t)$	medida de costo complejidad de un nodo no terminal t
$\mathfrak{L}$	conjunto de datos para construir modelo
$\mathfrak{X}$	conjunto de datos para predecir su variable objetivo
$\mathfrak{V}$	conjunto de índices del conjunto $\mathfrak{X}$
f_c	función de combinación en una unidad para una red neuronal artificial
f_t	función de transferencia en la unidad de salida de una red neuronal artificial
ϕ_k	función de transferencia en la unidad k de la capa de en medio en una red neuronal artificial
b_k	parámetro de sesgo en la unidad k de la capa de en medio para una red neuronal artificial
b^*	parámetro de sesgo en la unidad de salida para una red neuronal artificial
w_k	peso asignados en la conexión de la unidad k en la capa de entradas a la capa de en medio
w_k^*	peso asignado en la conexión de la unidad k en la capa de entradas hacia la capa de salida
$w_k^\star$	peso asignado en la conexión de la unidad k en la capa de en medio hacia la capa de salida
λ	parámetro para el decaimiento de los pesos en una red neuronal artificial
E	error por mínimos cuadrados en una red neuronal artificial
u	función de salida de una máquina de vectores de soporte
$\mathfrak{D}$	espacio del conjunto de entrenamiento para una máquina de vectores de soporte
$\mathfrak{H}$	espacio de mayor dimensión para mapear a $\mathfrak{D}$ de manera inplícita
ξ_i	variable que corrige el fallo del margen
$\Phi(X)$	función que mapea a X a $\mathfrak{H}$
$k(X, X')$	función que tienen la información de $< \Phi(X), \Phi(X') >$
a_i	multiplicadores de Lagrange para el problema de optimización en las máquinas de vectores de soporte

Símbolo	Significado
γ	variable que penaliza el error al limitar el efecto de los vectores de soporte
ν	parámetro para controlar el volumen de la esfera que separa a los datos extraños en una máquina de vectores de soporte
ρ	término que indica el error empírico del margen
Υ	eficiencia de un algoritmo
$\mathcal{A}$	conjunto de reglas de un algoritmo A
c_i	costo del paso i de un algoritmo
φ_k	modelo cuadrático que ayuda a aproximar la matriz Hessiana para la iteración k
$\triangle$	Simbolo Q. E. D. para indicar la finalización de una demostración.

OBJETIVOS

General

Plantear y, cuando sea posible, predecir situaciones de carácter crediticio en Guatemala mediante el uso de algoritmos y sistemas computacionales.

Específicos

1. Presentar tres algoritmos utilizados para predecir fenómenos, los cuales son: árboles de decisión, redes neuronales artificiales y máquinas de vectores de soporte.

2. Modelar la situación de crédito en una institución financiera en el periodo de un año por medio de dichos algoritmos.

3. Comparar los resultados obtenidos por el modelo con los reales para determinar la efectividad de los algoritmos.

4. Confrontar la efectividad de los tres métodos propuestos.

INTRODUCCIÓN

En este trabajo de graduación se tratarán propiedades de diversos algoritmos de agrupación, clasificación o predicción con el objetivo de analizarlos para posteriormente compararlos en un ambiente aplicado a la caracterización y predicción del estado de crédito de un grupo de personas. El crédito lo podemos ver como un medio de intercambio de bienes o servicios en donde una entidad le otorga un bien a otra, la cual se compromete en devolverla añadiendo un valor agregado en un determidado tiempo o una determinada situación. Debido a la naturaleza del compromiso entre ellos y el que las situaciones que se proyectan no siempre suceden como se pensaron es que para este crédito hay dos posibles resultados: que se complete el procedimiento o que se falle en el compromiso de devolver el bien como fue pactado; en caso que se falle el compromiso la entidad que otorga el bien pierde de alguna manera ese bien y en la manera de lo posible quiere evitar que suceda. Y para evitar que suceda es que surgieron técnicas para dar con cierta confianza y de una manera categórica la probabilidad de que se dé o no el fallo.

El cuerpo de este trabajo consta de tres partes principales: la primera siendo la presentación propia de tres de estos algoritmos, junto con su análisis computacional y descripción del método que se usará; la segunda para presentar los datos a analizar junto con la manera que se agruparon, particionaron y manipularon para la adaptación a los algoritmos; y la última parte, la de presentación de resultados y análisis de éstos, para hacer una comparación entre los métodos utilizados.

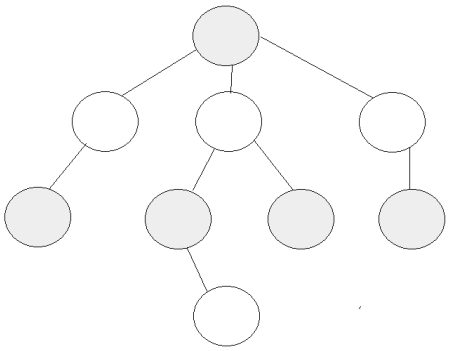
1. CONCEPTOS PRELIMINARES

1.1. Árboles de decisión

1.1.1. Noción

Los árboles de decisión son una herramienta muy útil que por mucho tiempo ha sido utilizada, siendo un ejemplo que encierra el concepto de los llamados árboles familiares, que de una manera sencilla lo que hacen es tomar a una persona como progenitor del árbol y en base a esa persona marcar las relaciones de «ser hijo de» y colocarlos respecto a cierto nivel que indica la cantidad de relaciones progenitor/hijo que hay de la persona al progenitor del árbol. Un ejemplo es presentado gráficamente en la figura 1.1. Al progenitor le llamaremos nodo raíz, a los últimos nodos en el árbol los nodos hoja y a los intermedios, nodos intermedios.

Figura 1.1. Ejemplo de la estructura gráfica de árbol.



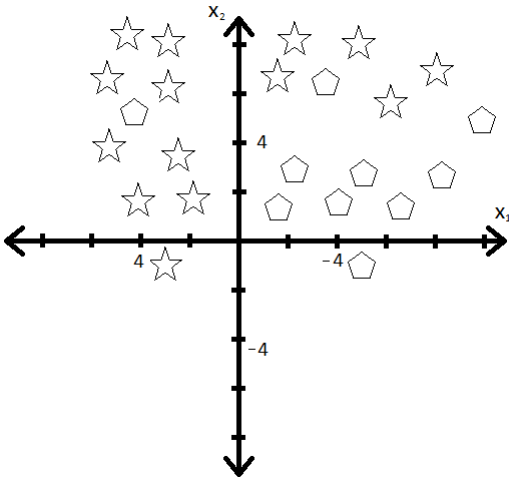
Fuente: elaboración propia, paint 6.2.

Como se comentó antes la idea es simple, predecir la respuesta de una variable Y la cual en un caso de clasificación es conveniente sea categórica, es decir que tenga una cantidad finita de valores que pueden ser etiquetados, por medio de otras variables x_1, x_2,..., x_n. Un método para realizarlo es haciendo crecer un árbol binario, es decir que cada nodo tiene como máximo dos hijos uno izquierdo y

otro derecho. En este árbol que crece desde el nodo raíz se evalúa una observación $X = (x_1, x_2, \ldots, x_n)$ y desde el nodo raíz se le hace una pregunta para determinar si pertenece al lado izquierdo o al lado derecho del nodo raíz, donde pertenecer significa que la respuesta a la pregunta fue *sí* para el lado izquierdo y *no* para el derecho, y así dicha observación se traslada al siguiente nodo dejando información de que estuvo en el nodo raíz; posteriormente en el siguiente nodo se realiza el mismo procedimiento para determinar el camino a seguir de la observación y se sigue haciendo este procedimiento hasta que se llegue a un punto en el cual el nodo ya no tiene hijos, es decir hasta que se llegue a un nodo hoja. Haciendo esto para todos los datos X_k que se tienen se obtiene sobre el árbol ya creado la clasificación que buscamos puesto que los nodos hojas son clasificados según las categorías de Y, empleando el procedimiento de separación que se da en el árbol. Con el propósito de ejemplificar la información de un árbol de clasificación se presenta el siguiente ejemplo:

Supongamos tenemos 31 observaciones sobre un fenómeno y que tiene dos posibles clases (siendo 19 de una clase y 12 de otra clase) y para las cuales existen dos variables numéricas que lo identifican. En la figura 1.2 las clases están marcadas como pentágonos o estrellas y las variables numéricas son x_1, x_2.

Figura 1.2. Ejemplo de distribución.

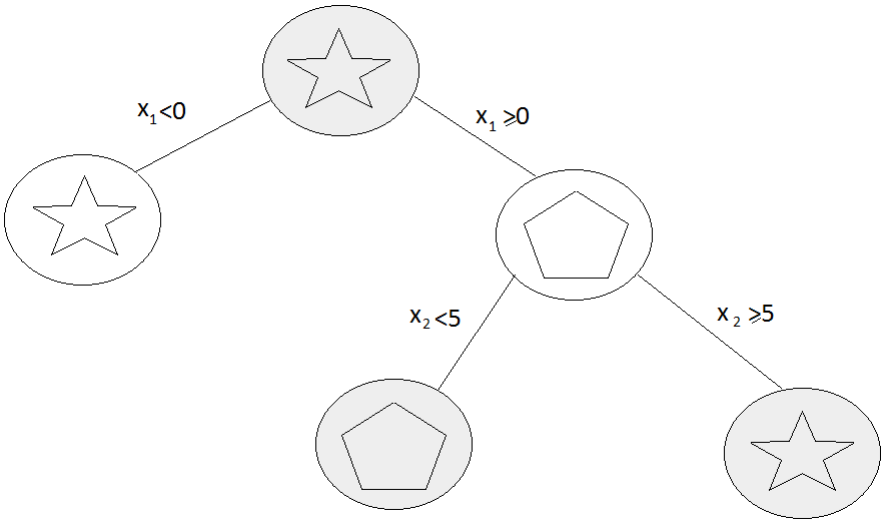


Fuente: elaboración propia, paint 6.2.

Observamos que del lado negativo del eje x_1 principalmente hay estrellas, mientras que en el positivo se nota una separación perpendicular al eje x_2 a la altura 5. Lo cual genera tres áreas sobre el plano x_1x_2 donde hay una clase predominante. Este procedimiento indica cómo realizar un árbol de clasificación para las 31 observaciones, siendo el primer paso separar las que tienen valor negativo (lado izquierdo

del árbol) de las de valor positivo o cero (lado derecho del árbol) para la variable x_1 y luego notamos que el lado izquierdo ya tiene en su mayoría una clase por lo que lo dejamos así y se vuelve un nodo hoja por otro lado el lado derecho tiene otra posible partición indicada por el tener valor menor (lado izquierdo del nodo) o mayor o igual a 5 (lado derecho del nodo) donde ya ambos nodos son nodos hoja y termina el procedimiento de clasificación. Visto dicho procedimiento gráficamente tenemos la figura 1.3.

Figura 1.3. Árbol de clasificación para el ejemplo planteado



Fuente: elaboración propia, paint 6.2.

En la figura 1.3 a cada nodo se le inscribió una figura: estrella o pentágono, simbolizando la clase mayoritaria presentada en cada nodo, dicho detalle ayuda a saber que si tenemos una observación que pasó por cierto nodo entonces la figura indica la clase a la que con más probabilidad dicha observación pertenezca.

1.1.2. Cómo crear un árbol

Para generar árboles de decisión hay diversos algoritmos para tener en consideración, ellos tienen algo en común que es la noción previamente descrita y se enriquecerá en este apartado. Con el propósito de mantener una estructura para clasificar a las personas que si completaron el contrato de crédito y a las que no, se tomó en consideración el tener solamente árboles binarios, por lo que solo de éstos se hablará. El procedimiento básico es repetitivamente particionar los datos en agrupaciones de observaciones más pequeñas de tal forma que los siguientes nodos hijos

(o la siguiente generación de ese nodo) tengan una mayor concentración de la clase prevaleciente en cada uno en comparación a la concentración en el nodo padre, es decir que en cada nodo hijo se tenga una mayor certeza de la clase a etiquetar sobre la certeza en el nodo padre; la forma en que se realiza la partición es por medio de una condición lógica en el nodo para una de las variables en cuestión, y del valor de verdad depende si se pertenece al lado izquierdo o al lado derecho del nodo.

El algoritmo a implementar está basado en el llamado **CART** que es un acrónimo de *Classification and Regression Trees*, publicado en 1984 por Leo Breiman, Jerome Friedman, Richard Olsen y Charles Stone en 1984. El algoritmo en resumen lo que hace es crear árboles con un procedimiento de partición, para luego tomar subárboles por otro método que descarta nodos, y finaliza comparando dichos subárboles para escoger el mejor.

Dado que el árbol a fin de cuentas nos da una regla de clasificación de los datos, antes de iniciar el algoritmo debe de considerarse que de los datos presentes no todos deben ser usados en la creación (usualmente llamado entrenamiento) del árbol, puesto que el objetivo de hacer dicho árbol va más allá de usarse con los datos con los que se creó sino para usar las reglas de asignación para clasificar un conjunto de datos distinto, para así predecir o comparar está clasificación.

Por lo que antes de iniciar la creación del árbol debemos preparar bien nuestros datos en tres conjuntos: uno para el entrenamiento, otro para la prueba de la efectividad del árbol, y el tercero que es el objetivo del algoritmo (los datos que se desean clasificar).

La variable sobre la que se realiza una partición depende de la optimización de una función de pureza o impureza que indica cuán buena es la partición. Para hablar de dicha función y su posterior aplicación necesitamos darle variables a la información del árbol. Provisto que este procedimiento se aplicará a un nodo y a sus dos hijos y es independiente de los demás nodos en el sentido que una vez la información haya llegado a ese nodo no hay forma en que se mueva de ahí, entonces podemos usar variables de manera local y que se repitan para cada nodo. Así sean t_p el nodo padre, t_i y t_d los nodos hijos izquierdo y derecho respectivamente, x_j la variable que será la indicadora de la partición y x_j^* el valor o categoría respecto de cual se realizará la partición.

Definición 1.1. Una **función de impureza** para un árbol de decisión es una función i la cual asocia a cada nodo t un valor real sobre un intervalo $[a, b]$ (usualmente $[0, 1]$) que indica algún tipo de proporción de una clase presente en el nodo sobre el total de las observaciones presentes en el nodo (la homogeneidad del nodo).

Limitándonos al caso donde la variable que particionará los datos es numérica, cuando se realiza una partición, la impureza del nodo padre t_p se toma constante, ya que se asume el valor que dio mejor desempeño en la partición anterior (en caso sea el nodo raíz el resultado es el mismo), para todos los posibles valores de corte. Así la máxima homogeneidad de los nodos hijos es equivalente a la diferencia de las impurezas.

Definición 1.2. La **diferencia de impurezas** para un árbol de decisión es una función Δi la cual asocia a cada nodo t_p y a sus dos nodos hijos t_i y t_d (abreviados con t_c) el valor de la impureza del padre substrayéndole la media ponderada de las impurezas de los nodos hijos, la cual se puede escribir de la siguiente manera:

$$\Delta i(t_p) = i(t_p) - E[i(t_c)]. \tag{1.1}$$

Sean P_i, P_d las probabilidades de pertenecer a los lados izquierdo y derecho del nodo padre respectivamente, es decir la proporción de los datos en cada nodo hijo respecto a la cantidad de datos del nodo padre. Entonces podemos ver a la diferencia de impurezas y posteriormente al problema de optimización como:

$$\Delta i(t_p) := i(t_p) - P_i i(t_i) - P_d i(t_d).$$

$$\underset{x_j \leq x_j^*, j=1,\ldots,n}{\arg\ \max} [i(t_p) - P_i i(t_i) - P_d i(t_d)]. \tag{1.2}$$

Este problema de optimización no debe de considerarse que busca a x_j^* entre todos los valores reales sino que busca entre los valores de los datos de entrenamiento, los cuales pueden ser muchos pero son finitos, por lo que no se necesita analizar en sí la optimización ya que solo lucha con el tiempo de la máquina y da la garantía de ser el máximo.

En el caso que alguna de las variables no sea numérica sino categórica, el análisis es algo diferente, se define una función de impureza, su diferencia y si se considera particionar sobre dicha variable entonces se hace la partición para cada una de las clases que tiene la variable categórica, generando así un conjunto de nodos hijos de misma cardinalidad que clases en la variable categórica y al conjunto de ellos lo denotaremos como tt_c.

Ya que tenemos la forma de realizar las particiones con la impureza solo falta definir la impureza a utilizar, y hay más de una posible dependiendo de salida (la

variable a predecir y). En el caso de que sea continuo se puede utilizar la minimización de la suma residual de cuadrados, pero como este no es el caso no se comentará más. Como la situación que tenemos es de respuesta binaria, sí o no al completar el trato, los tipos de impureza propuestos son: el **índice Gini**[1], la **entropía** y el **error de clasificación**; los cuales dependen del factor $\boldsymbol{p}$ definido como la proporción en cada nodo de la clase minoritaria.

Como la impureza a utilizar en este documento es el índice Gini se tomará especial consideración explicándolo y se define como:

$$i_G(t) := 2p(1-p). \tag{1.3}$$

El índice Gini representa la suma de las multiplicaciones de las proporciones de una clase con la proporción de cada una de las otras clases; dando así la probabilidad de que dos elementos escogidos aleatoriamente de la misma población sean de la misma clase. Como se tienen dos clases presentes hay dos sumandos y son iguales debido a simetría; y se busca minimizarlo para tener una clase con proporción considerablemente mayor, ya que vista como función de p el índice es una parábola con coeficiente principal negativo y con máximo en el punto $(0.5, 0.5)$ y como su dominio es $[0.5, 1]$ buscamos el mínimo de i que se da cuando p es mayor. Otra forma de ver el índice es por el lado de la **pureza** (que es el opuesto a la impureza en donde buscaríamos maximizarla), donde la pureza la define como la suma de los cuadrados de las proporciones de las clases.

Para la entropía la impureza se define como el valor negativo de la suma de las multiplicaciones de las proporciones de las clases con el logaritmo base 2 de dichas proporciones como sigue:

$$i_E(t) := -p\log_2(p) - (1-p)\log_2(1-p). \tag{1.4}$$

Por otro lado para el error de clasificación errónea la impureza se define al indicar la proporción de la clase minoritaria:

$$i_e(t) := 1 - \text{máx}(p, 1-p). \tag{1.5}$$

Además de estos cuatro métodos mencionados, existen otros con el mismo objetivo y efectividad similar. Por lo que no es muy significativo cual de todos escoger ya que no es en esto donde radica la efectividad del algoritmo, ésta radica

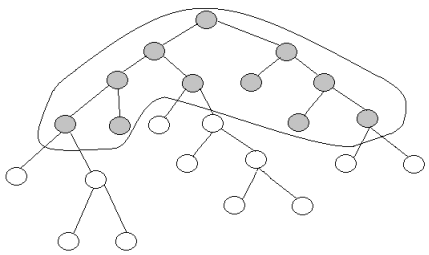
[1]En honor al estadístico italiano Corrado Gini.

principalmente en el método de poda del árbol.

El procedimiento de hacer particiones recursivamente se puede hacer hasta que se alcanza cierto criterio de parada, el cual puede ser: un límite para la impureza, o equivalentemente una pureza alcanzada, o una cantidad mínima de observaciones en un nodo para particionarlo. En el caso del límite de la impureza, cuando en un nodo se tiene una impureza menor a dicho límite se detiene el proceso de hacer particiones; en el caso de pureza alcanzada cuando en un nodo se tiene una pureza mayor a dicho límite se detiene el prodedimiento; en el caso de la cantidad mínima de observaciones en un nodo para particionarlo, cuando en un nodo hay menos de dicha cantidad entonces el procedimiento de particionar se detiene. Cuando en un nodo se llega al criterio de parada, entonces el nodo se considera nodo hoja. Sin embargo aplicar dichos criterios suena simple pero puede ser complicado, ya que si un nodo luego de una partición que se considera pobre y según los criterios ya no debería de particionarse sobre él puede que si tenga en unos niveles de profundidad más un corte que sea considerable. Por ello la estrategia es dejar que crezca el árbol sin límite y usar otro procedimiento para mejorar el árbol.

Una vez se ha construido el árbol dejando en los nodos hojas la mínima impureza que se podía por medio de particiones recursivas, ya tenemos clasificada a la muestra de entrenamiento, pero puede que la tengamos demasiado clasificada y que nuestro árbol solo se ajuste a ella, por lo tanto debe de haber algún procedimiento que diga cuándo nuestro árbol está sobre-ajustado a los datos de entrenamiento. Dicho procedimiento lo conocemos como **poda**. El procedimiento de poda para el algoritmo CART busca las ramas del árbol que tienen menos poder predictivo por hoja, lo cual lo logra al considerar al **costo por complejidad**, que es una medida que impone una penalización por la complejidad debida a la cantidad de nodos hoja en el subárbol. Así, aquellas ramas que tienen probabilidad alta de sobre-ajustarse a los datos de entrenamiento son penalizadas y marcadas para la poda.

Figura 1.4. Ejemplo de subárbol obtenido mediante la poda



Fuente: elaboración propia, paint 6.2.

Observemos que las ramas de por sí también tienen la estructura de un árbol si le incluimos al nodo padre que se convierte en nodo hoja con la poda, y así son subárboles. Por notación a los primeros árboles creados los denotaremos por T_0 y a sus subárboles con T. Así sea $T \subset T_0$ un subárbol que se va a podar, si m la usamos de índice para los nodos hojas del subárbol T, a N_m la cantidad de observaciones en el nodo hoja m y $\overline{T}$ el conjunto de nodos hoja en el subárbol T (al cual en sí le podemos decir subárbol), entonces la definición del criterio de costo complejidad es:

$$C_\alpha(T) := \sum_{m=1}^{|\overline{T}|} R_{t_m} + \alpha|\overline{T}| \tag{1.6}$$

donde α es el parámetro de complejidad y R_{t_m} es un estimador del error local de T para el nodo indexado con m, usualmente se usa el error de clasificación del nodo por la probabilidad de que una observación quede en dicho nodo; si N es la cantidad de observaciones en los datos de entrenamiento y t_m denota al nodo etiquetado con m, R_{t_m} queda dado por:

$$R_{t_m} := \frac{i_e(t_m)N_m}{N} \tag{1.7}$$

Notemos que la definición de R_{t_m} puede generalizarse para cualquier nodo t. El término $\sum_{m=1}^{|\overline{T}|} R_{t_m}$ donde los nodos t_m son nodos hojas del árbol T, es la **estimación global del error de la clasificación** y la podemos abreviar con la notación $R(T)$.

El objetivo es encontrar el subárbol T_α que minimice el costo complejidad $C_\alpha(T)$. Para el valor $\alpha = 0$ se tiene el árbol completo, mientras que mayores valores de α dan diversos árboles de menor complejidad por nodos (de menos nodos terminales), mientras que el caso extremo para cierto valor de α_K $K \in \mathbb{N}$ en el que la penalización por complejidad prevalece a la medida global del error de la clasificación, ya que $\alpha \geq 0$ entonces puede crecer tanto como se quiera, ya que T es constante alcanzando así que T_{α_K} sea solo el nodo raíz. Por lo cual, el valor α es importante en la compensación entre la complejidad del subárbol y su ajuste a los datos, y además se tiene que existe una sucesión finita de subárboles obtenida a partir de los valores α por los que el árbol minimizado cambia (T_{α_0},T_{α_1},...,T_{α_K}). Si t_0 es el nodo raíz se tiene:
$T_0 \supseteq T_{\alpha_1} \supset ... \supset T_{\alpha_k} = t_0$
Para obtener esta sucesión de subárboles se procede como sigue:

a) Obtener el árbol de partida T_{α_1} que debe ser el subárbol más pequeño posible cuya medida global del error de la clasificación sea igual que la de T_0. El cual

puede ser el mismo que T_0 en caso que no haya habido un criterio de parada y cada nodo terminal tenga impureza cero.

En caso que haya habido criterio de parada se debe podar todo par de nodos terminales t_i indexado con m_i y t_d indexado con m_d que comparten como padre inmediato al nodo t_p que se le asigna el índice m_* (suponiendo t_p podría reemplazar a sus hijos como nodo terminal) y cumplan $R_{t_{m_*}} = R_{t_i} + R_{t_d}$ y podarlos a todos. El subárbol generado después de la poda es T_{α_1}.

De esta forma se cumple que para el árbol T_{α_1} se da que $R_t > R(T_t)$ donde t es un nodo no terminal de T_{α_1} y T_t es un subárbol que tiene al nodo t como nodo raíz y contiene a todos los descendientes de t en T_0, ya que (considerando a t como t_p) se da $R(t_p) \geq R(t_i) + R(t_d)$ puesto que R se ve afectada junto con la impureza y si se sigue desarrollando la misma desigualdad sobre los nodos hijos hasta que se llegue a los nodos hojas tenemos ya que siempre se da que $R(t_p) \geq R(T_{t_p})$ pero no se puede dar la igualdad porque de los nodos hoja de T_0 fueron podados los que cumplían la igualdad (de haber existido) para construir a T_{α_1}.

b) Obtener el parámetro α_2 y su respectivo subárbol. Para esto empezamos considerando los nodos no terminales t y les definimos medidas de costo complejidad como sigue:

$$C_\alpha(t) := R(t) + \alpha. \tag{1.8}$$

Con esto para $\alpha = 0$ se tiene que $C_\alpha(t) > C_\alpha(T_t)$ (es una conclusión del paso anterior). Como ambos lados de la desigualdad dependen de α pero la cantidad menor tiene a α multiplicado por un natural distinto de cero, entonces si aumenta el valor de α, C_α crece más por lo que se da un punto en el cual $C_\alpha(t) = C_\alpha(T_t)$ así tenemos:

$$R(t) + \alpha = R(T_t) + \alpha|T_t| \Rightarrow \alpha = \frac{R(t) - R(T_t)}{|T_t| - 1}. \tag{1.9}$$

Como la medida de coste complejidad es una ayuda para el procedimiento de poda, lo usamos con el objetivo de ir podando las ramas más débiles primero, así sea el nodo t_1 cuyos hijos son la rama más débil encontrado como el nodo que minimice el valor de α obtenido de la expresión anterior (ecuación 1.9) ya que menor valor de α más cercanis están los estimadores de errores

de clasificación, siendo así los más probables a sobre-ajustar el modelo a los datos de entrenamiento. Así construimos el árbol T_{α_2} podando de T_{α_1}, T_{t_1}, lo cual lo podemos ver como:

$$T_{\alpha_2} := T_{\alpha_1} - T_{t_1}. \tag{1.10}$$

Donde $\alpha_2 := \min_{t \in T_{\alpha_1}} \frac{R(t) - R(T_t)}{|T_t| - 1}$.
Aquí notemos que la poda vista como diferencia de árboles no quita al nodo raíz de la rama.

c) Repetir el paso dos de manera recursiva, hasta que se lleve al límite de los α, es decir cuando se llegue a un α tal que $T_\alpha = t_0$ (el cual es denotado con α_K).

Ahora nuestra sucesión de subárboles sólo debemos escoger a uno, el que tenga el menor error. Puesto que cada uno tiene distinta penalización con sobre-ajustarse a los datos de entrenamiento. Para determinar dicho árbol no podemos compararlo con la efectividad que tiene sobre el conjunto de entrenamiento ya que caeríamos en un absurdo al querer comparar que nuestro subárbol no se ajusta a los datos de entrenamiento al ver como se ajusta a ellos, por lo que aquí es donde necesitamos al subconjunto de datos de validación para escoger el que tenga el menor estimador de error de clasificación (o podría medirse en cuanto a la efectividad del árbol con otros métodos) con estos datos. Una vez hemos escogido al subárbol de menor error, debemos ponderar la efectividad de éste con el conjunto de prueba.

1.2. Redes neuronales artificiales

1.2.1. Noción

Las redes neuronales artificiales son un tipo de herramienta que puede ser aplicada en varios campos y tiene una buena medida de éxito al aplicarse correctamente. El detalle particular de las redes neuronales artificiales es que buscan emular las redes neuronales biológicas al modelar en una computadora las conexiones neuronales del cerebro humano, y tienen de alguna forma la capacidad de aprender de los datos.

La forma en la que las redes neuronales artificiales emulan (o lo intentan) a una red neuronal biológica es en la parte que el ser vivo mediante sus sentidos percibe las cosas a su alrededor y para que el ser realice una actividad tienen que darse ciertas condiciones, así cada sentido busca el cómo están dichas condiciones y manda una

lista de respuestas a la solicitud de percepciones con la cual las neuronas por medio de una combinación de las percepciones decide si realizar o no la actividad.

Así una red neuronal artificial es una colección de unidades (neuronas) distribuidas en tres grupos o capas uno de entrada uno intermedio y uno de salida. En el de entrada es donde se definen las variables que utilizará la red para pasarlas a la capa intermedia, en el intermedio es donde ocurren los procesos de evaluar las entradas y devolver una conclusión a la capa de salidas, y en el de salida se combinan los resultados previos para dar una salida. Pero para hacer esto al igual que en los árboles de decisión se necesita un conjunto de datos que le ayude a saber cómo realizar las combinaciones, es el equivalente al aprendizaje de la vida en los seres vivos.

El resultado del aprendizaje por medio del entrenamiento son pesos distribuidos por toda la red. Sin embargo el motivo por el que fueron escogidos los pesos no es reportado y en contraste con el listado de reglas del árbol de decisión las redes neuronales no reportan el motivo de lo que hacen, aunque no por ello se ha de pensar que hacen un procedimiento aleatorio para ello sino que matemáticamente hablando fueron escogidos porque hacen que el método dé mejores resultados; de manera similar en una red neuronal a veces el ser vivo realiza actividades por impulso sin tener una noción muy clara de la razón.

Las redes neuronales son buenas para problemas de predicción y estimación. Siempre y cuando el que las utilice tenga claro lo siguiente:

a) Los datos con los que se construirán las redes neuronales. Saber qué variables son las que realmente importan aunque no se sepa cómo combinarlas.

b) La variable a predecir. Es necesario saber qué es lo que se trata de modelar.

c) Es necesario darle mucha experiencia a la red neuronal artificial. Es influyente la cantidad de datos con los que entrenará la red.

1.2.2. Cómo construir una red

Una vez se tenga el conocimiento sobre los datos, la variable a predecir y una cantidad suficiente de observaciones registradas, el procedimiento para construir el modelo se podría describir con los siguientes pasos:

a) Transformar los datos de entrada al rango $(-1, 1)$

La importancia de esto radica en los pesos y el que se escogen de manera similar para cada entrada, por lo que los requerimientos para las entradas deben se ser similares en este aspecto del rango y que sea pequeño para que a

la red neuronal artificial se le haga más fácil encontrar patrones. Para esto se considera si los datos son numéricos o categóricos y en base a ello se escoge el método para utilizarlos en la red neuronal artificial.

Si los datos son numéricos y continuos, como por ejemplo cantidades monetarias o medidas físicas, como dichos valores no son infinitos, entonces son un conjunto acotado y caen en un rango teniendo su valor mínimo y valor máximo. Con estos valores se puede hacer un cambio de escala a $(-1, 1)$ con una trasformación lineal. Se tiene la ventaja que las imágenes mantienen la misma distribución que los datos iniciales, pero se tiene la desventaja de que si los datos están sesgados, al hacer la transformación puede que la diferencia entre los nuevos datos sea considerable pero no realmente significativa, sin embargo la red neuronal le daría importancia.

Por lo que es mejor considerar otro método siendo uno de los posibles la estandarización de la variable, es decir sustraer la media y dividir entre la desviación estándar, pero en este método se debe de tener cuidado con los valores que están muy lejos ya que al igual que antes tienen repercusión, en este caso le dan más peso a la desviación estándar.

Si los datos son numéricos pero en un orden discreto con números enteros, como por ejemplo cantidad de objetos, edad y categorías ordenadas, para esto una solución sencilla es contar cuántos valores diferentes tienen los datos y particionar el intervalo $(0, 1)$ en esa cantidad de datos y asignar a las variables su fracción proporcional, por ejemplo si hay 3 valores distintos entonces según su orden se les asignan los valores $0, 1/2, 1$.

Otro método que es útil cuando el salto de 0 a 1 es significativo pero de los valores mayores o incluso de 1 en adelante no tanto es el llamado *código de termómetro* que básicamente cuenta la cantidad de valores en la variable y luego a cada valor se le asigna un número binario de longitud igual a la cantidad de valores en la variable (con ceros incluidos en cualquier espacio) por medio de la siguiente regla: según el orden del valor (el cual empieza desde cero) se le asigna un 1 en la posición de dicho orden del número binario (contado de izquierda a derecha) y en todas las posiciones anteriores a ésta. Luego todos los números binarios son divididos entre el número binario con una unidad más de dicha longitud con un solo uno en la primer posición (de izquierda a derecha); por ejemplo si hay 3 valores distintos se les asigna los valores $000_2/1000_2 = 0$, $100_2/1000_2 = 4/8 = 1/2$ y $110_2/1000_2 = 6/8 = 3/4$

respectivamente.

Si los datos son de índole categórica, como por ejemplo datos demográficos, podemos considerar hacer lo mismo que en el caso de números enteros ordenados y partir el intervalo, aunque en algunos casos el orden que se le asigne no tiene importancia pero la red así lo puede tomar. Otra forma es, en lugar de usar dicha variable, utilizar banderas al aumentar el número de variables por la cantidad de valores categóricos y asignar el valor 1 cuando el valor de la variable es el valor de la bandera y 0 cuando no, y aunque se aumenten las variables se elimina la necesidad de asignar un orden que puede no representar algo real.

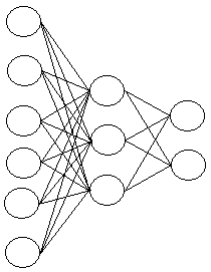
Si los datos no caen en ninguno de estos tipos, debe de estudiarse la naturaleza de los datos y buscar aplicar alguna técnica como las que se usaron anteriormente en los otros casos.

b) Diseñar la estructura de red correcta

Como ya se dijo antes la red consta de tres grupos de unidades, y la forma en la que éstos se relacionan es llamada la **topología** de la red. En dicha topología la capa intermedia no es imprescindible, mientras que las otras dos capas si lo son. En caso de utilizar unidades en la capa intermedia (el caso más usual) el número de ellas puede variar conforme se vean los resultados, considerando que mientras más de ellas más patrones se encuentran y mientras más se tengan el riesgo a sobre-ajustarse crece.

La capa de salida puede tener más de una unidad de salida. Algunas veces es conveniente permitir que la capa de salidas se conecte con la capa de entradas y así considerar que con dichas conexiones se hacen regresiones que la capa intermedia ajusta.

Figura 1.5. Ejemplo de red neuronal artificial



Fuente: elaboración propia, paint 6.2.

Un ejemplo de topología de red se presenta en la figura 1.5, que tiene 6 unidades en la capa de entradas, 3 en la capa intermedia y 2 en la capa de salida.

El sistema de transferencia de la red se llama de alimentación hacia adelante cuando la información se envía en una sola vía que es de las unidades de entradas a las intermedias y luego a las de salida.

c) Entrenar la red

Para entender cómo funciona el entrenamiento hay que tener claro lo que hace cada neurona artificial con los datos que recibe, primero los combina en un valor el cual es manipulado para dar un valor de salida; al procedimiento de combinar, transformar para mandar un resultado se le llama **función de activación**, que es una función que se mantiene con una salida no considerable hasta que la combinación alcance cierto límite en el cual ya toma más importancia (y se considera activada). En la función de activación están dos procedimientos o funciones que son la función que hace propiamente la combinación y la función que toma dicho valor y lo transforma o transfiere a otro valor para dar de salida; estás funciones se llaman **función de combinación** y **función de transferencia** denotadas como f_c y f_t respectivamente.

Un ejemplo de función de combinación es la suma con pesos, donde cada variable tiene su peso así su valor se multiplica por dicho peso y se suman todos los valores así obtenidos, otras posibles funciones de combinación son el máximo o mínimo de las entradas ya pesadas. En este caso usaremos la suma con pesos; la podemos ver de la siguiente manera: Como las observaciones son de la forma $X = (x_1,\ x_2,...\ x_{n^*})$, para n^* variables, los pesos para dichas variables son $w = (w_1, w_2, ..., w_{n^*})$ y si b es el sesgo (o parámetro que indica cuán fácil es activar dicha unidad) de la unidad entonces:

$$f_c(X) := X \cdot w + b. \tag{1.11}$$

Entre las posibles funciones de transferencia se tienen a la logística[2], lineal y tangente hiperbólica[3]. Estás funciones nos ayudan a aplicar cierto modelo de regresión a los datos que fueron combinados por unidad. El uso de funciones

[2]Una función de la forma $\dfrac{1}{1+e^{-x}}$.

[3]Una función de la forma $\dfrac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$.

lineales equivale a la regresión lineal mientras que la forma de S que presentan las funciones logísticas y la tangente hiperbólica le dan al modelo propiedades no lineales y éstas se diferencian en el rango ya que la logística va de 0 a 1 y la hiperbólica de -1 a 1. Así las redes neuronales artificiales son buenas en el modelaje de problemas lineales (con la función lineal), problemas casi lineales (con las funciones en forma de S y cerca de cero con forma casi lineal) y los problemas no lineales (con las funciones en forma de S lejos de cero). Y combinaciones de estos ya que cada unidad puede tener una función de transferencia diferente, para así tener la posibilidad de resolver un problema más complejo.

Además de los pesos asignados para cada variable en la función de combinación, la función de transferencia tiene también un peso por unidad en la que vive, por ejemplo si la función de transferencia se decide será función logística tenemos:

$$f_t(Z) := \frac{1}{1 + e^{-Z}}. \tag{1.12}$$

Donde Z es el valor que f_c le da al vector X. Notemos que si Z es un valor positivo grande entonces $e^{-Z} \approx 0$ y $f_t(Z) \approx 1$ y si Z es un valor negativo de gran magnitud entonces e^{-Z} crece y $f_t(Z)$ se acerca a cero.

El objetivo de las funciones f_c y f_t (que no deben verse por separados sino que en conjunto), es mover los datos desde el valor de salida cero hasta el valor uno y la variedad de las formas de hacer dicho recorrido, por ello se menciona la forma de S, así la salida y activación de la unidad dependerá de la *suavidad* en la que el valor de $X \cdot W + b$ se mueva en la función de transferencia (vista gráficamente).

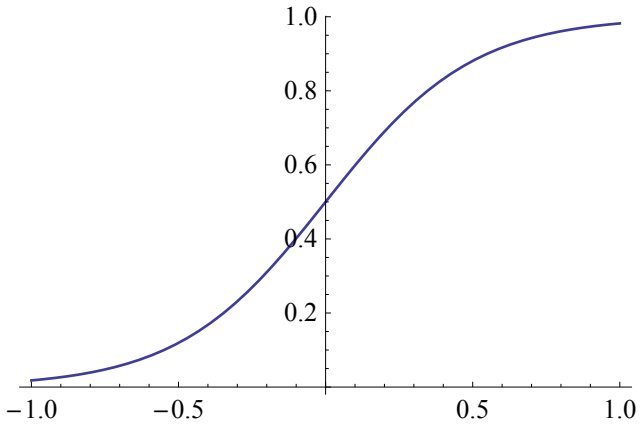
Definición 1.3. La **suavidad** de una función de activación es como pequeños cambios en los pesos y en el sesgo producen pequeños cambios en la salida. Así del cálculo tenemos:

$$\Delta\text{salida} = \sum_{j=1}^{n^*} \frac{\partial Y}{\partial w_j} \Delta w_j + \frac{\partial Y}{\partial b} \Delta b. \tag{1.13}$$

Entonces como estas funciones tienen la característica de variar su activación por los pesos, se puede prever como el cambio de los pesos afecta la salida.

Así entrenar una red es proveer a la red de los datos de entrenamiento para que realice iterativamente múltiples regresiones y las combine para asig-

Figura 1.6. Forma de S para una función de transferencia



Fuente: elaboración propia, Wolfram Mathematica 9.

nar los pesos a las variables que hagan que la combinación de las regresiones tenga menos error. Sea el conjunto de entrenamiento de M observaciones tomadas del conjunto $\mathfrak{L} = \{(X_n, Y_n) \mid n = 1, 2, ..., N; X_n \in \mathbb{R}^{a+b}, \}$ para $X_n = (x_1^n, x_2^n, ..., x_a^n, x_{a+1}^n, ..., x_{a+b}^n)$ donde $x_1, x_2, ..., x_a$ son variables numéricas elegidas en $\mathbb{R}$ y $x_{a+1}^n, x_{a+2}^n, ..., x_{a+b}^n$ son la representación numérica de variables categóricas que viven en el espacio $\mathbb{E}$ de todas las clases posibles entre todas las variables categóricas, considerar aquí que todos estos datos ya fueron transformados al intervalo $(-1, 1)$ y que el conjunto $\mathfrak{L}$ se reordenó de tal forma que los índices de los M elementos seleccionados para entrenamiento son los primeros M.

Ya con la función de activación f como combinación de f_c y f_t, por notación la función de transferencia en la unidad k de la capa intermedia la escribiremos como ϕ_k y a su sesgo como b_k; y a la función de transferencia de la unidad de salida como f_t con su sesgo b^*. Además la unidad de entradas k tiene su peso w_k cuando envía los datos a la capa intermedia y w_k^* cuando se envía a la capa de salida, y para la unidad intermedia k tiene su peso $w_k^\star$. Así tenemos que el resultado en la red neuronal para la observación X está dada en la ecuación 1.14 y lo denotaremos como y^*, si la cantidad de unidades en la capa intermedia es h,. Y adicionalmente a la función que termina realizando la red neuronal la denotaremos con f_n, así f_n tiene la siguiente forma:

$$f_n(X) = f_t(b^* + \sum_{i=1}^{a+b} w_i^* x_i + \sum_{j=1}^{h} w_j^\star \phi_j(b_j + \sum_{i=1}^{a+b} w_i x_i). \tag{1.14}$$

Las redes neuronales artificiales cumplen que en la capa de salidas la función de transferencia lineal pueden aproximar uniformemente cualquier función continua en un conjunto compacto al aumentar la cantidad de unidades en la capa intermedia.

Definición 1.4. Como **criterio de adaptación en una red** utilizamos el de suma de mínimos cuadrados E, el cual se resume en la siguiente fórmula:

$$E := \sum_{i=1}^{M} |Y_i - f_n(X_i)|^2. \tag{1.15}$$

La utilidad de utilizar esta función en lugar del error de clasificación es que para el error de clasificación, pequeños cambios en los pesos pueden darnos el mismo resultado en la clasificación pero sí nos da una ligera variación en el costo. Primero se encuentran los pesos que minimizan el costo para después poder utilizar el error de clasificación para medir el rendimiento de la red.

Un método para garantizar que la función f_n sea suave es la *regularización* que penaliza a la función, es decir le aumenta a su error una función de penalización multiplicada por λ, un parámetro que indica cuán fuerte es la penalización (e implícitamente indica cuán fuerte debe ser el cambio en los pesos). Dicha función de penalización específica de las redes neuronales es la suma de los cuadrados de los pesos. Así definimos la siguiente función:

$$C(f_n) := \sum_{i=1}^{a+b} (w_i^2 + (w_i^*)^2) + \sum_{j=1}^{h} (w_j^\star)^2. \tag{1.16}$$

Quedando así el criterio de acoplamiento como:

$$E + \lambda C(f_n). \tag{1.17}$$

El uso de este descenso en el peso ayuda penalizando la red para evitar sobreajuste a los datos de entrenamiento y adicionalmente ayuda a la optimización de los pesos. La optimización de los pesos se hace por medio del método **BFGS**[4] (Broyden, Fletcher, Goldfarb y Shanno) que es un método cuasi-Newton conocido también como algoritmo de métrica variable que usa los valores de la función y el gradiente para hacer una imagen de la superficie a

[4]Método del cual no se comentará mucho ya que no es el objetivo del trabajo

ser optimizada.

d) Validar la red

Puesto que cuando se entrena la red siempre se tiene la posibilidad de sobreajuste a los datos de entrenamiento, siempre es necesario darle cierta validez a la red para garantizar que haya aprendido de los datos de entrenamiento pero que no los haya memorizado y sólo en ellos funcione. Esto se consigue al utilizar la red en los datos de validación y dependiendo del error que se dé, se toma como una buena red o no.

e) Evaluar la red

Una vez se haya validado como buena una red ésta se debe de dar como resultado juntamente con su error, pero ni el error sobre los datos de entrenamiento ni el error sobre los datos de validación es significativo ya que éstos fueron utilizados para escoger la red por lo que están diseñados para tener un buen resultado sobre dichos datos, por lo que se necesita un tercer conjunto de datos para calcular el error o eficiencia que presenta la red.

1.3. Máquinas de vectores de soporte

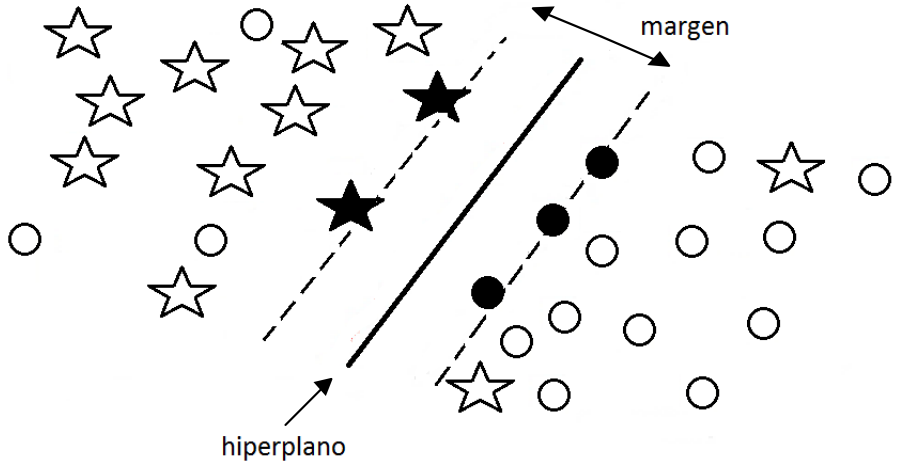
1.3.1. Descripción

Las máquinas de vectores de soporte son algoritmos de clasificación y con ello predicción sobre una variable binaria o de solamente dos clases, como lo es el problema a tratar en este trabajo. Básicamente este algoritmo busca el óptimo hiperplano con máximo *margen* que separe a ambas clases en el espacio $\mathbb{R}^n$ donde pertenecen las variables numéricas que caracterizan a los elementos del conjunto a clasificar, al que llamaremos espacio de entradas. Entonces las máquinas de vectores de soporte buscan un método lineal para separar los datos que pertenecen al espacio de entradas y si no encuentra el separador lineal, proyecta los datos en un espacio de mayor dimensión donde sí se da la separación lineal, la proyección la realiza por técnicas *kernel*.

Definición 1.5. El **margen** de un hiperplano separador es la distancia más corta del hiperplano a un elemento de cada clase de las que separa, a las que llamaremos **vectores de soporte**. Lo cual se puede visualizar en la figura 1.7, donde el

hiperplano separa a los elementos de las dos clases (círculos y estrellas), el margen es la distancia entre el hiperplano y los elementos más cercanos a éste, que son los vectores de soporte.

Figura 1.7. Ejemplo de máquina de vectores de soporte



Fuente: elaboración propia, paint 6.2.

La función de clasificación o salida de una máquina de vectores de soporte en el caso más simple, es decir el caso lineal se describe a continuación:

Suponiendo tenemos N observaciones (X_i, Y_i) que tienen variables de valores numéricos o categóricos, $X_i = (x_1^i, x_2^i, ..., x_n^i)$ donde $x_1, x_2, ..., x_n$ son variables numéricas que pertenecen a $\mathbb{R}$ y Y_i es la variable con dos posibles resultados convertidos a 1 y -1. Entonces la separación del espacio $\mathbb{R}^n$ se da por un funcional lineal f de tal forma que define los subespacios:

$$A_1 := \{Z \in \mathbb{R}^n | f(z) < b\} \qquad A_2 := \{Z \in \mathbb{R}^n | f(z) > b\}$$

Y como $\mathbb{R}^n$ es un espacio de Hilbert, el funcional tiene una representación de Riesz y $f(X) = \langle X, w \rangle$ con $w = (w_1, w_2, ..., w_n)$. Así la máquina de vectores de soporte tiene como aplicación de salida, para X en el conjunto de entradas, la siguiente función:

$$U(X) = \sum_{i=1}^{n} x_i w_i - b \tag{1.18}$$

Donde el hiperplano separador es aquél que cumple $U = 0$, que denotaremos por L y los puntos más cercanos caen en el hiperplano $U = \pm 1$. Entonces el margen es la distancia equivalente a esa unidad 1 de distancia entre los valores de las imágenes de

U, que corresponde a la distancia entre los vectores de soporte y el hiperplano. Para ello consideremos que el vector w es perpendicular al plano ya que para z_1, $z_2 \in L$:

$\langle z_1 - z_2, w\rangle = \langle z_1, w\rangle - \langle z_2, w\rangle$ y como $\langle z_1, w\rangle + b = 0$ y $\langle z_2, w\rangle + b = 0$ entonces $\langle z_1, w\rangle - \langle z_2, w\rangle = -b + b = 0$.

Entonces $z_1 + \dfrac{w}{\|w\|_2}$ es un elemento que está a 1 unidad (según la métrica usual en $\mathbb{R}^n$) del hiperplano separador, por lo que el valor de ese elemento evaluado en U está a una diferencia de:

$$u(z_1 + \frac{w}{\|w\|_2}) = \langle z_1 + \frac{w}{\|w\|_2}, w\rangle - b = \langle z_1, w\rangle + \langle \frac{w}{\|w\|_2}, w\rangle - b = \|w\|_2.$$

Así que los elementos que cumplen con $U = \pm 1$ están a $\dfrac{1}{\|w\|_2}$ de distancia del hiperplano separador ya que la diferencia del valor de u con la distancia sobre w de dos vectores son cantidades proporcionales, por lo que el margen m está dado por:

$$m = \frac{2}{\|w\|_2}. \tag{1.19}$$

Las máquinas de vectores de soporte buscan maximizar dicha expresión (o el cuadrado de la misma) para así tener el hiperplano que separe dejando la mayor distancia posible entre las clases, es decir el detalle de ellas radica en el problema de optimización siguiente:

$$\underset{b,w}{\textbf{mín}} \frac{1}{2}\{\|w\|_2\} \qquad \text{s.a.} \quad Y_i(w \cdot X_i - b) \geq 1, \quad \forall i = 1, 2, 3, ..., N. \tag{1.20}$$

Aunque esta optimización se llevaría a cabo de una manera más eficiente con solamente los vectores de soporte, debido a que todavía no se ha entrenado a la máquina, no se puede saber cuáles serían vectores de soporte por lo que los datos X son cualesquiera para el conjunto de entrenamiento. Para resolver el problema se considera la solución por la lagrangiana y con ello construir el problema dual obtenido al optimizar la expresión para los multiplicadores de Lagrange a. Ya que en el óptimo global las condiciones de Kuhn-Tucker garantizan que el gradiente de la lagrangiana es cero, y así convirtiendo el problema de optimización al problema dual por medio

de la lagrangiana tenemos:

$$\begin{aligned} \min_{a} & \left\{ \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} a_i a_j Y_i Y_j (X_j \cdot X_i) - \sum_{i=1}^{N} a_i \right\} \\ \text{s.a.} \quad & \sum_{i=1}^{N} a_i Y_i = 0, a_i \geq 0 \quad \forall i = 1, 2, ..., N. \end{aligned} \tag{1.21}$$

Además en el procedimiento se obtiene que $w = \sum_{i=1}^{N} a_i Y_i X_i$ y para algún $a_k > 0$ se tiene que $b = w \cdot X_k - y_k$. El manipular el problema es útil ya que baja el tiempo computacional para encontrar w vía la solución del problema dual de optimización.

Por otro lado, como no todos los conjuntos pueden ser linealmente separables entonces se modifican los argumentos originales a optimizar (ecuación 1.20) y en lugar de solamente optimizar el margen, se le agrega una penalización que a la vez permite a un elemento del conjunto de entrenamiento pertenecer al lugar que debería debido a su clasificación, ésta modificación es la siguiente:

$$\begin{aligned} \min_{b,w} & \frac{1}{2} \left\{ \|w\|_2 + \gamma \sum_{i=1}^{N} \xi_i \right\} \\ \text{s.a.} \quad & Y_i(w \cdot X_i - b) \geq 1 - \xi_i, \ \xi_i \geq 0 \quad \forall i = 1, 2, 3, ..., N. \end{aligned} \tag{1.22}$$

Donde γ es la variable que penaliza la máquina de vectores de soporte al limitar el efecto de los vectores de soporte sobre la función de salida y ξ_i son variables que corrigen el fallo del margen para cada observación en el conjunto de entrenamiento, es decir la variable que acerca a los elementos en el lugar donde deberían estar por su clasificación. Al realizar esta modificación, el problema dual queda similar solamente con el cambio de que ahora $\gamma \geq a_i \geq 0 \quad \forall i = 1, 2, ..., N$.

Así tenemos totalmente determinado el modo operacional de las máquinas de vectores de soporte para el caso en que se puede encontrar en el espacio de entradas un hiperplano separador, lo cual sucede pero no siempre, por lo que ya entendiendo el caso sencillo procedemos a analizar algo más general, que requiere el utilizar un mapeo para las entradas y así trasladarlos a un espacio de más dimensiones donde sí pueda ocurrir la separación. El resultado es similar con la salvedad de que cambia los vectores de entradas por su mapeo al espacio de más dimensiones, y ya con estas nuevas entradas se hace el mismo procedimiento que el ya realizado; el detalle de esto es que primero se debe decidir qué mapeo utilizar y luego calcular los mapeos del conjunto de entrenamiento para luego poder hacer los productos

internos con vectores de mayor dimensión, y así no presenta mayor complejidad en el razonamiento, sólo en el cómputo de los mapeos y los productos internos, lo cual si es considerable ya que encontrar la forma del mapeo no es sencillo. Por estas consideraciones las máquinas de vectores de soporte utilizan un procedimiento analítico para evitar utilizar dicho mapeo, el cual es el siguiente:

Si Φ es el mapeo que manda al conjunto de entrenamiento (al que denotaremos como $\mathfrak{D}$) al espacio de mayor dimensión, entonces se considera definido de manera implícita por una función kernel, es decir, una función cuya entrada son los vectores del conjunto de entrenamiento y su salida es el producto interno de las imágenes de dichos puntos en el mapeo Φ. Así si $\Phi : \mathfrak{D} \to \mathfrak{H}$ con $\mathfrak{H}$ el espacio de mayor dimensión, entonces la función kernel $k : \mathfrak{D} \times \mathfrak{D} \to \mathbb{R}$ para $X, X' \in \mathfrak{D}$ está dada por:

$$k(X, X') := \langle \Phi(X), \Phi(X') \rangle \tag{1.23}$$

Esta técnica es usualmente llamada como el «truco del kernel» y ayuda computacionalmente hablando ya que es más sencillo que mapear explícitamente los vectores X, X' en $\mathfrak{H}$. Lo que se debe de considerar ahora es escoger el kernel correcto o diseñarlo para así trabajar en espacios de gran dimensión de manera implícita y computacionalmente más sencilla, siendo la salida de la máquina la siguiente:

$$u(X) = \Phi(X) \cdot w - b. \tag{1.24}$$

Entonces se plantea el mismo problema pero con condiciones diferentes, quedando así el sigiuente problema de optimización:

$$\begin{aligned} &\min_{b,w} \frac{1}{2} \left\{ \|w\|_2 + \gamma \sum_{i=1}^{N} \xi_i \right\} \\ &\text{s.a.} \quad Y_i(w \cdot \Phi(X_i) - b) \geq 1 - \xi_i, \ \xi_i \geq 0 \ \ \forall i = 1, 2, 3, ..., N. \end{aligned} \tag{1.25}$$

Donde el vector w y la solución final, es decir la función salida de la máquina de vectores de soporte, se encuentran de manera parecida al caso no lineal debido a que el método a aplicar aquí es el mismo, quedando así de la siguiente manera:

$$w = \sum_{i=1}^{N} a_i Y_i \Phi(X_i), \tag{1.26}$$

$$u(X) = \sum_{i=1}^{N} Y_i a_i k(X, X_i) - b. \tag{1.27}$$

Donde los valores $a_i \ \forall \ i = 1, 2, ..., N$ están dados por el problema:

$$\begin{aligned} \min_{a} & \left\{ \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j Y_i Y_j k(X_j, X_i) - \sum_{i=1}^{N} a_i \right\} \\ \text{s.a.} \quad & \sum_{i=1}^{N} a_i Y_i = 0, \gamma \geq a_i \geq 0 \ \ \forall i = 1, 2, ..., N. \end{aligned} \tag{1.28}$$

El cual es resuelto con optimización cuadrática, siendo un problema QP que tiene una complejidad cuasipolinomial[5].

Ahora bien, si en lugar de clasificar lo que queremos es encontrar casos extraños en los datos, lo que se hace es de alguna forma con una máquina de vectores de soporte crear un tipo de esfera en la que los datos que caen afuera son los datos a considerar como extraños, lo cual se logra con el parámetro ν que modifica el volumen de la «esfera» que deja afuera a los vectores extraños al establecer una cota superior para la fracción de extraños a encontrar y ρ es un término que indica el error empírico del margen, en este caso la optimización de la función de salida de la máquina de vectores de soporte es:

$$\begin{aligned} \min_{b,w,\rho} & \frac{1}{2} \left\{ \|w\|_2 + \frac{1}{\nu} \sum_{i=1}^{N} \xi_i - \rho \right\} \\ \text{s.a.} \quad & (w \cdot \Phi(X_i) - b) \geq \rho - \xi_i, \ \xi_i \geq 0 \ \ \forall i = 1, 2, 3, ..., N. \end{aligned} \tag{1.29}$$

Donde de igual manera se construye el problema dual y se encuentran los valores de los multiplicadores de Lagrange con la ayuda del truco del kernel.

1.3.2. Posibles funciones kernel

Para escoger la función a usar como kernel se puede construir una en base a los datos y la experiencia adquirida por las máquinas de vectores de soporte o en el caso más general se puede escoger entre las siguientes:

a) El kernel lineal

$$k(X, X') := \langle X, X' \rangle$$

[5]Es decir que el tiempo que se utiliza en su solución crece más rápido que de manera polinomial pero más lento que de manera exponencial, dependiendo de la cantidad de variables

b) El kernel de base radial Gaussiana

$$k(X, X') := e^{-\sigma\|X-X'\|^2}$$

c) El kernel polinomial

$$k(X, X') := (\text{escala}\langle X, X'\rangle) + \text{sesgo})^{\text{grado}}$$

d) El kernel de tangente hiperbólica

$$k(X, X') := \tanh(\text{escala}\langle X, X'\rangle + \text{sesgo})$$

e) El kernel de base radial Laplaciana

$$k(X, X') := e^{-\sigma\|X-X'\|}$$

Los kernel de base radial son usados generalmente cuando no se tiene un gran conocimiento previo sobre los datos, el lineal cuando los datos son vectores dispersos, el polinomial cuando se trata de procesamiento de imágenes y el de tangente hiperbólica para el cálculo en redes neuronales.

Para el entrenamiento de las máquinas de vectores de soporte las variables necesitan ser numéricas por lo que se debe hacer, al igual que para las redes neuronales, una conversión de las categorías, en el caso más sencillo y rápido de determinar por medio de banderas. Además también se consideran los conjuntos de datos para entrenamiento, para validación y para prueba. El primer conjunto para la creación de la función y optimización de la misma, el segundo para encontrar los parámetros óptimos del modelo y el de prueba, para dar el error o eficiencia de la máquina.

2. ANÁLISIS DE ALGORITMOS

Definición 2.1. Un algoritmo es un conjunto ordenado de reglas a seguir para solucionar un problema, el cual puede ser visto como asociación de entradas con salidas. Por lo que el algoritmo busca construir las salidas del problema al recibir las entradas. Los aspectos a resaltar de un algoritmo entonces son los siguientes:

a) *Entradas.* Los datos que el problema le da al algoritmo para realizar cierto procedimiento.

b) *Salidas.* Los datos que el problema espera que devuelva el algoritmo al realizar cierto procedimiento.

c) *No ambiguo.* Que el algoritmo siempre tiene presente qué comando utilizar o regla a seguir.

d) *Finito.* Que el algoritmo tiene una cantidad finita de reglas.

e) *Correcto.* Que el algoritmo realice el procedimiento que se le otorga.

f) *Efectividad.* Que cada instrucción del algoritmo pueda llevarse a cabo.

g) *General.* Que el algoritmo debe ser capaz de resolver el problema para cualquier conjunto de entradas posible.

Un algoritmo siempre debe de ser *correcto*; mientras que el que sea claro, el que esté bien estructurado, el que sea amigable con el usuario y el que sea eficiente son aspectos extra que tienen su importancia pero no son estrictamente necesarios.

Definición 2.2. La **eficiencia** de un algoritmo se mide en términos de la cantidad de recursos de cómputo que requiere, los cuales es tiempo de ejecución o **costo en tiempo** y la cantidad de memoria que utiliza o **costo en espacio**.

Por lo que se busca que un algoritmo correcto sea eficiente, es decir que mantenga lo más bajo posible los costos de tiempo y espacio en comparación a otros

algoritmos correctos que hagan la misma tarea, por ello la comparación en eficiencia de dos algoritmos correctos para la misma tarea se trata de que uno es más eficiente que el otro si utiliza menos recurso en tiempo y espacio.

Para hacer comparaciones entre algoritmos se puede recurrir a realizar pruebas empíricas y medir el coste que cada algoritmo tiene para compararlos, pero no es lo mejor, ya que depende mucho de las condiciones en las que se realizan las pruebas, por lo que lo mejor es realizar un análisis matemático de los algoritmos para que con dicho análisis ya se pueda comparar de una manera más significativa.

Para realizar el análisis se debe de contar la cantidad de operaciones significativas en cada paso para luego expresar la eficiencia de los algoritmos usando el crecimiento de funciones.

Como ya se comentó, un algoritmo consta de pasos ordenados, pero cada regla puede ser de distinta naturaleza dependiendo de las operaciones fundamentales que realiza, dichos pasos pueden ser de las siguientes categorías según su naturaleza:

a) **Definición**. Es el tipo de paso cuyo objetivo es asignar un valor a cierta variable o a ciertas variables. El costo de la definición es constante para todas las posibles definiciones en el paso, y la cantidad de veces que se hace el paso es la cantidad de veces que se realizarán las definiciones.

   La estructura del paso para definición es la siguiente:

DEFINICIÓN	*Costo*	*# veces*
1: $variable_1 \leftarrow valor_1$	c_1	1

b) **Condición**. Es el tipo de paso cuyo objetivo es hacer una comparación que devuelve un valor lógico y en base al valor lógico realizar un procedimiento u otro. Las comparaciones tienen un costo que puede diferir según el tipo de comparación y la cantidad de veces que se realiza depende de la estructura sobre la cual se realice la comparación.

   El caso más sencillo de una condición es cuando se realiza una comparación, la cual es una estructura *si* que hace la comparación y si el valor lógico es verdadero entonces realiza el procedimiento. Entonces el algoritmo y su análisis son:

CONDICIÓN SIMPLE	*Costo*	*# veces*
1: **si** comparación **entonces**	c_1	1
2: Procedimiento	c_2	1

En el caso que se tenga un procedimiento a realizar tanto cuando la condición es verdadera como cuando es falsa entonces el algoritmo y su análisis son:

CONDICIÓN SIMPLE CON CASO FALSO	*Costo*	*# veces*
1: **si** comparación **entonces**	c_1	1
2: Procedimiento si verdadero	c_2	1
3: **si no**	c_3	1
4: Procedimiento si falso	c_4	1

En el caso más general cuando se tienen varias condiciones para realizar varios procedimientos el algoritmo y su análisis son:

CONDICIÓN GENERAL	*Costo*	*# veces*
1: **si** comparación 1 **entonces**	c_1	1
2: Procedimiento si condición 1	c_2	1
3: **si no si** comparación 2 **entonces**	c_3	1
4: Procedimiento si condición 2	c_4	1
5: **si no**	c_5	1
6: Procedimiento si falso todo	c_6	1

c) **Ciclo**. Es el tipo de paso que realizará cierto procedimiento tantas veces como se indique acorde a ciertas condiciones. Los ciclos que se usarán son el ciclo *mientras* y *para*.

En el caso de *mientras* el procedimiento se realizará tantas veces la condición se cumpla, por lo que se sugiere que en el procedimiento se cambien las variables a usar en la siguiente condición, para garantizar que no sea un ciclo infinito. El procedimiento y análisis para este ciclo es:

CICLO *mientras*	*Costo*	*# veces*
1: **mientras** comparación	c_1	t
2: Procedimiento	c_2	$t-1$

Donde t es la cantidad de veces que se realizará la condición y el procedimiento se realiza $t-1$ veces ya que la última comparación fue la que detiene ciclo.

En el caso de *para* el procedimiento se realiza la cantidad de veces según el conjunto de los valores especificados para el índice, ya que no hay condición a cumplir. El procedimiento y análisis para este ciclo son:

Ciclo *para*	*Costo*	*# veces*
1: **para** índice ∈ conjunto	c_1	$\|conjunto\| + 1$
2: Procedimiento	c_2	$\|conjunto\|$

Donde la cantidad de veces que se realiza la comparación en el ciclo es $|conjunto|+1$ puesto que hay una comparación en la que el ciclo se da cuenta que ya cubrió todos los índices del conjunto.

Para esto se tienen las siguientes consideraciones:

a) Cada paso tiene su costo, el cual será denotado como c_i donde i es el paso sobre el que se analiza el costo.

b) Para que un algoritmo entre a un ciclo (*mientras* o *para*) se realiza una prueba de entrada, además las pruebas son contadas en el paso que indica el entrar al ciclo. Es importante considerar para que el ciclo termine de manera normal es necesario que realice una prueba más.

c) Para los ciclos adentro de un ciclo es importante considerar que cada iteración del ciclo de adentro puede llevarse a cabo una cantidad de veces diferente de las demás dependiendo de condiciones en las iteraciones del ciclo que lo contiene.

Así podemos ver a la eficiencia o coste como una función Υ sobre el algoritmo A, según sus entradas $\mathcal{A}$, que asigna la sumatoria sobre todos los pasos a llevarse a cabo, el producto del costo del paso con la cantidad de veces que se hace el paso.

Cuando se hace el análisis del algoritmo se colocan las cantidades de veces que el algoritmo realizará dicho paso pero es una cantidad desconocida por lo se debe de considerar las cantidades representativas del peor caso, del caso intermedio y del mejor caso.

Pero realizar dicha sumatoria para todos los pasos del algoritmo completo puede ser algo tedioso y muy largo por lo que en lugar de considerar la eficiencia exacta se usa un concepto que de alguna manera abstrae la información de la eficiencia. Dicho concepto es la *razón de crecimiento* o el *orden de crecimiento*.

Definición 2.3. La **razón de crecimiento** o el **orden de crecimiento** de un algoritmo es la razón a la que el coste de un algoritmo crece cuando la cantidad o el tamaño de las entradas crece. Es decir si n es la cantidad de elementos en $\mathcal{A}$ entonces si la eficiencia del algoritmo es proporcional a n se dice que tiene un orden

de crecimiento lineal, así si el crecimiento está dado por un polinomio de grado k entonces su orden de crecimiento es de n^k. Donde siempre el término de la eficiencia cuyo crecimiento es más significativo con gran cantidad de datos es el que domina al orden, realizando un análisis asintótico.

Con el concepto de orden de crecimiento: un algoritmo es más eficiente que otro cuando su orden de crecimiento es asimptóticamente menor al del otro algoritmo y no sólo cuando su costo es menor.

Definición 2.4. Según las características de las razones de crecimiento para una función $g(n)$ se tienen las siguientes definiciones:

a) $\Theta(g(n)) := \{f(n)$: existen constantes positivas $c_1, c_2 \in \mathbb{R}$ y $n_0 \in \mathbb{N}$ tales que $0 \leq c_1\ g(n) \leq f(n) \leq c_2\ g(n)$ cuando $n \geq n_0\ \}$.

b) $\Omega(g(n)) := \{f(n)$: existen constantes positivas $c \in \mathbb{R}$ y $n_0 \in \mathbb{N}$ tales que $0 \leq c\ g(n) \leq f(n)$ cuando $n \geq n_0\ \}$.

c) $O(g(n)) := \{f(n)$: existen constantes positivas $c \in \mathbb{R}$ y $n_0 \in \mathbb{N}$ tales que $0 \leq f(n) \leq c\ g(n)$ cuando $n \geq n_0\ \}$.

d) $o(g(n)) := \{f(n)$: para todo positivo $c \in \mathbb{R}$ existe $n_0 \in \mathbb{Z}^+$ tal que $0 \leq f(n) \leq c\ g(n)$ cuando $n \geq n_0\ \}$.

e) $\omega(g(n)) := \{f(n)$: para todo positivo $c \in \mathbb{R}$ existe $n_0 \in \mathbb{Z}^+$ tal que $0 \leq c\ g(n) \leq f(n)$ cuando $n \geq n_0\ \}$.

Entonces expresaremos la eficiencia de un algoritmo en base a cualquiera de estas cuatro razones de crecimiento y haremos comparaciones en base a ellas. El análisis se hace por partes al dividir el algoritmo en subprocesos para analizarlos ya sea uno por uno o de manera recursiva y luego se hace una suma más sencilla usando la notación anterior.

2.1. Algoritmo para árbol de decisión

2.1.1. Pseudocódigo

Supongamos que tenemos N observaciones (X_n, Y_n) que tienen variables de valores numéricos o categóricos, $X_n = (x_1^n, x_2^n, ..., x_a^n, x_{a+1}^n, ..., x_{a+b}^n)$ donde $x_1, x_2, ..., x_a$

son variables numéricas que pertenecen a $\mathbb{R}$ y $x_{a+1}^{n}, x_{a+2}, ..., x_{a+b}$ son variables categóricas que viven en el espacio $\mathbb{E}$ de todas las clases posibles entre todas las variables categóricas; entonces queremos construir un árbol de clasificación para predecir y comparar el comportamiento de un conjunto de observaciones $\mathfrak{X} = \{(X_v, Y_v)\}_{s\in\mathfrak{v}}$ distintas a las N, dicho árbol tendrá un criterio de parada en la partición de nodos dado por el parámetro β, que indica la diferencia en impureza mínima que debe de poder crear un nodo con su mejor partición, y además se tomará una proporción de M/N datos para el conjunto de entrenamiento dejando a los restantes de los N para la validación. Entonces el pseudocódigo que usaremos como ejemplificación para crear un árbol de decisión es:

Algoritmo 1 Árbol de decisión

Entrada: $\mathfrak{L} = \{(X_n, Y_n)|n = 1, 2, ..., N; X_n \in \mathbb{R}^a \times \mathbb{E}^b, \}$
Parámetro de partición β
Cantidad de elementos en el conjunto de entrenamiento M
$\mathfrak{X} = \{(X_v, Y_v\}_{s\in\mathfrak{v}}$

Procedimiento CREAR EL ÁRBOL (Usando muestra de M elementos de $\mathfrak{L}$)
    $T \leftarrow \{t_0\}$
    $\overline{T} \leftarrow \{t_0\}$
    $cont \leftarrow 0$
    mientras $|T| > cont$
        $cont \leftarrow |T|$
        para t $\in \overline{T}$
            para m de 1 a $a + b$
                si $m > a$ **entonces**
                    Calcular Δi
                    $\Delta i_m \leftarrow \Delta i$
                si no
                    Guardar los V valores que x_m toma en el nodo t en el conjunto: $\{u_1, u_2, ..., u_V\}$ con la condición que $u_j < u_{j+1}$ para $j \in \{1, 2, ..., V - 1\}$
                    para j de 1 a $V - 1$
                        Calcular Δi cuando $x_m \leq u_j$
                        $\Delta i_{m_j} \leftarrow \Delta i$
                    Encontrar el valor de j que maximiza el valor de Δi_{m_j} para $j \in \{1, 2, ..., V - 1\}$
                    Guardar dicho valor en j^*
                    $vc_m \leftarrow \dfrac{u_{j^*} + u_{j^*}}{2}$
                    Calcular Δi con la condición $x_m \leq vc_m$
                    $\Delta i_m \leftarrow \Delta i$

Encontrar el valor de m que maximiza el valor de Δi_m para $m \in \{1, 2, ...a + b\}$
Guardar dicho valor en m^*
si $\Delta i_{m^*} \geq \beta$ **entonces**
  si $m^* > a$ **entonces**
    Hacer la partición acorde a las clases de la variable x_{m^*} y construir el conjunto tt_c
    $T \leftarrow T \cup tt_c$
    $\overline{T} \leftarrow \overline{T} \cup tt_c \setminus \{t\}$
  si no
    Hacer la partición acorde al corte en el valor vc_{m^*} y obtener los nodos t_d y t_i
    $T \leftarrow T \cup \{t_i, t_d\}$
    $\overline{T} \leftarrow \overline{T} \cup \{t_d, t_i\} \setminus \{t\}$

Procedimiento PODAR EL ÁRBOL (Con los datos de entrenamiento)
  $l \leftarrow 1$
  $T_{\alpha_1} \leftarrow T$
  $\overline{T}_{\alpha_1} \leftarrow \overline{T}$
  para $t \in \overline{T}_{\alpha_1}$
    si t_p es el nodo padre de t y (t_i, $t_d \in \overline{T}_{\alpha_1}$ o si $tt_c \subset \overline{T}_{\alpha_1}$) **entonces**
      si $R(t_p) = R(t_i) + R(t_d)$ **entonces**
        $T_{\alpha_1} \leftarrow T_{\alpha_1} \setminus \{t_i, t_d\}$
        $\overline{T}_{\alpha_1} \leftarrow \overline{T}_{\alpha_1} \cup \{t_p\}$
  mientras $T_{\alpha_l} \neq \{t_0\}$
    para $t \in T_{\alpha_l} \setminus \overline{T}_{\alpha_l}$
      $T_t \leftarrow$ el conjunto de todos los nodos de la rama de t
      $\overline{T}_t \leftarrow$ el conjunto de todos los nodos hojas de la rama del nodo t
      $\alpha_{m_2} \leftarrow \dfrac{R(t) - R(T_t)}{|\overline{T}_t|}$
      $t_{m_2} \leftarrow t$
      $m_2 \leftarrow m_2 + 1$
    Encontar el valor de m_2 que minimiza el valor de α_{m_2} para $m_2 \in \{1, 2, ...|t \in T_{\alpha_l} \setminus \overline{T}_{\alpha_l}|\}$
    Guardar dicho valor en m_2^*
    $T_{\alpha_{l+1}} \leftarrow T_{\alpha_l} - T_{t_{m_2^*}}$
    $\overline{T}_{\alpha_{l+1}} \leftarrow \overline{T}_{\alpha_l}\ \overline{T}_{t_{m_2^*-1}}$
    $\overline{T}_{\alpha_{l+1}} \leftarrow \overline{T}_{\alpha_{l+1}} \cup \{t_{m_2^*-1}\}$
    $l \leftarrow l + 1$

23: **Procedimiento** VALIDAR EL ÁRBOL (Con los datos de $\mathfrak{L}$ que no fueron usados en el entrenamiento)
24: **para** $j\{\in \{1, 2, ..., l-1\}$
25: $err_j \leftarrow R(T_{\alpha_j})$
26: Encontrar el valor de j que minimiza el valor de $R(T_{\alpha_j})$ for $j \in \{1, 2, ..., l-1\}$
27: Guardar dicho valor en j^*

1: **Procedimiento** PROBAR EL ÁRBOL(Con los datos de $\mathfrak{X}$)
2: $Err \leftarrow R(T_{\alpha_{j^*}})$

Salida: El mejor árbol es $T_{\alpha_{j^*}}$ con un error global de estimación de Err.

2.1.2. Análisis del algoritmo

Teorema 2.1.1. *El costo en tiempo de ejecución del algoritmo 1 tiene una razón de crecimiento de* $\Theta(N^4)$ *en el peor caso.*

Demostración: Para analizar el algoritmo empecemos considerando los procedimientos descritos uno por uno:

CREAR EL ÁRBOL	*Costo*	*# veces*
1: $T \leftarrow \overline{T} \leftarrow \{t_0\}$	c_1	2
2: $cont \leftarrow 0$	c_2	1
3: **mientras** $\|T\| > cont$	c_3	N
4: $cont \leftarrow \|T\|$	c_4	$N-1$
5: **para** t $\in \overline{T}$	c_5	$\sum_{k=1}^{N-1} k$
6: **para** m de 1 a $a+b$	c_6	$\sum_{k=1}^{N-1}\sum_{l=1}^{k}(a+b+1)$
7: **si** $m > a$ **entonces**	c_7	$\sum_{k=1}^{N-1}\sum_{l=1}^{k}(a+b)$
8: $\Delta i_m \leftarrow \Delta i$	c_8	$\sum_{k=1}^{N-1}\sum_{l=1}^{k} b$
9: **si no**	c_9	$\sum_{k=1}^{N-1}\sum_{l=1}^{k}(a+b)$
10: $\{u_1, u_2, ..., u_V\}$	c_{10}	$\sum_{k=1}^{N-1}\sum_{l=1}^{k} a * \Theta(N^2)$
11: **para** j de 1 a $V-1$	c_{11}	$\sum_{k=1}^{N-1}\sum_{l=1}^{k} a * N$
12: $\Delta i_{m_j} \leftarrow \Delta i$	c_{12}	$\sum_{k=1}^{N-1}\sum_{l=1}^{k} a * (N-1)$

Crear el árbol *continuación*		*Costo*	*# veces*
13:	$j^* \leftarrow$ máx Δi_{m_j}	c_{13}	$\sum_{k=1}^{N-1}\sum_{l=1}^{k} a*(N-1)+1$
14:	$vc_m \leftarrow \frac{u_{j^*}+u_{j^*}}{2}$	c_{14}	$\sum_{k=1}^{N-1}\sum_{l=1}^{k} a$
15:	$\Delta i_m \leftarrow \Delta i,\ x_m \leq vc_m$	c_{15}	$\sum_{k=1}^{N-1}\sum_{l=1}^{k} a$
16:	$m^* \leftarrow$ argmáx Δi_m	c_{16}	$\sum_{k=1}^{N-1}\sum_{l=1}^{k} (a+b-1)+1$
17:	**si** $\Delta i_{m^*} \geq \beta$ **entonces**	c_{17}	$\sum_{k=1}^{N-1} k$
18:	**si** $m^* > a$ **entonces**	c_{18}	$\sum_{k=1}^{N-1} k$
19:	Particionar, construir tt_c	c_{19}	$\sum_{k=1}^{N-1} k$
20:	$T \leftarrow T \cup tt_c;\ \overline{T} \leftarrow \overline{T} \cup tt_c \setminus \{t\}$	c_{20}	$\sum_{k=1}^{N-1} k$
21:	**si no**	c_{21}	$\sum_{k=1}^{N-1} k$
22:	Particionar, $t_{d/i} \leftarrow \{t_d, t_i\}$	c_{22}	$\sum_{k=1}^{N-1} k$
23:	$T \leftarrow TUt_{d/i};\ \overline{T} \leftarrow \overline{T}Ut_{d/i} \setminus \{t\}$	c_{23}	$\sum_{k=1}^{N-1} k$

Podar el árbol		*Costo*	*# veces*
24:	$l \leftarrow 1$	c_{24}	1
25:	$T_{\alpha_1} \leftarrow T$	c_{25}	1
26:	$\overline{T}_{\alpha_1} \leftarrow \overline{T}$	c_{26}	1
27:	**para** $t \in \overline{T}_{\alpha_1}$	c_{27}	$N+1$
28:	**si** t_p y (t_i, $t_d \in \overline{T}_{\alpha_1}$ o si $tt_c \subset \overline{T}_{\alpha_1}$) **entonces**	c_{28}	N
29:	**si** $R(t_p) = R(t_i) + R(t_d)$ **entonces**	c_{29}	N
30:	$T_{\alpha_1} \leftarrow T_{\alpha_1} \setminus \{t_i, t_d\}$	c_{30}	N
31:	$\overline{T}_{\alpha_1} \leftarrow \overline{T}_{\alpha_1} \cup \{t_p\}$	c_{31}	N
32:	**mientras** $T_{\alpha_l} \neq \{t_0\}$	c_{32}	$N+1$
33:	**para** $t \in T_{\alpha_l} \setminus \overline{T}_{\alpha_l}$	c_{33}	$\sum_{k=1}^{N}(N+1)$

PODAR EL ÁRBOL *continuación*		*Costo*	*# veces*
34:	$T_t \leftarrow$ nodos de la rama de t	c_{34}	$\sum_{k=1}^{N} N$
35:	$\overline{T}_t \leftarrow$ nodos hojas de la rama del nodo t	c_{35}	$\sum_{k=1}^{N} N$
36:	$\alpha_{m_2} \leftarrow \frac{R(t) - R(T_t)}{\lvert\overline{T}_t\rvert}$	c_{36}	$\sum_{k=1}^{N} N$
37:	$t_{m_2} \leftarrow t$; $m_2 \leftarrow m_2 + 1$	c_{37}	$\sum_{k=1}^{N} N$
38:	$m_2^* \leftarrow$ mín α_{m_2}	c_{38}	$N+1$
39:	$T_{\alpha_{l+1}} \leftarrow T_{\alpha_l} - T_{t_{m_2^*}}$	c_{39}	N
40:	$\overline{T}_{\alpha_{l+1}} \leftarrow \overline{T}_{\alpha_l}\ \overline{T}_{t_{m_2^*-1}}$	c_{40}	N
41:	$\overline{T}_{\alpha_{l+1}} \leftarrow \overline{T}_{\alpha_{l+1}} \cup \{t_{m_2^*-1}\}$	c_{41}	N
42:	$l \leftarrow l + 1$	c_{42}	N

VALIDAR EL ÁRBOL		*Costo*	*# veces*
43:	**para** $j\{\in \{1, 2, ..., l-1\}$	c_{43}	$N+1$
44:	$err_j \leftarrow R(T_{\alpha_j})$	c_{44}	N
45:	Encontrar el valor de j que minimiza el valor de $R(T_{\alpha_j})$ for $j \in \{1, 2, ..., l-1\}$	c_{45}	N
46:	Guardar dicho valor en j^*	c_{46}	1

PROBAR EL ÁRBOL		*Costo*	*# veces*
47:	$Err \leftarrow R(T_{\alpha_{j^*}})$	c_{47}	1

De donde podemos ver que el costo en tiempo del algoritmo tiene la forma de $A + BN + CN^2 + DN^3 + EN^4 + FN^5$, donde A, B, C, D, E, F son constantes que dependen de las cosntantes c_i, con lo cual el término de mayor orden es el N^4 que corresponde a $c_{10} * \sum_{k=1}^{N-1} \sum_{l=1}^{k} a * \Theta(N^2) = a * \Theta(N^2) * \sum_{k=1}^{N-1} k = a * \Theta(N^2) * (N-1)(N)/2 = \Theta(N^4)$ en el algoritmo. $\triangle$

2.1.3. Software

El software que se utilizará para realizar la clasificación y predicción de datos es *R Console* versión 3.2.3. Y el paquete a utilizar es **rpart**. Para crear árboles se utilizó

el comando *tree*, el cual devuelve un objeto tipo *tree* que tiene las especificaciones de las particiones que fueron realizadas en su construcción, código que se usa de la siguiente manera:

```
rpart (formula, data, subset, method, control= tree.control(minsplit,
maxdepth,cp))
```

formula: expresión que indica la variable a predecir y las variables a utilizar. $Y \sim x_1 + x_2 + ... + x_n$.

data: tabla con las observaciones.

subset: arreglo de índices que indican los datos a utilizar de la tabla en data.

method: método usado para evaluar las particiones.

minsplit: cantidad mínima de observaciones que deben estar presentes en un nodo para que se intente hacer el corte sobre él.

maxdepth: la profundidad máxima que el árbol puede alcanzar.

cp: parmámetro del factor por el que cada corte a hacerce debe ser capaz de reducir al error de clasificación.

Para utilizar el mejor modelo obtenido en un nuevo conjunto de datos se hace de la siguiente manera, que devuelve un conjunto de datos donde a cada valor a predecir se le asigna el vector de probabilidades de pertenecer a tal clase o la clase con mayor probabilidad de ser:

```
predict (tree, newdata, type = c (“vector”, “class”))
```

tree: árbol obtenido con rpart.

newdata: conjunto de prueba para el árbol.

type: tipo de salidas entre vector de probabilidad o clases.

Para ver el error de clasificación se puede hacer de la siguiente manera, que devuelve una tabla donde se puede ver cuantos fueron clasificados correctamente y cuantos no lo fueron:

```
table (predict, newdata [, Y])
```

predict: predicción obtenida con predict de tipo clase.

newdata: conjunto de prueba del árbol.

Y: variable a predecir.

2.2. Algoritmo para redes neuronales

2.2.1. Pseudocódigo

Sean N observaciones (X_n, Y_n) que tienen variables numéricas o categóricas, $X_n = (x_1^n, x_2^n, ..., x_a^n, x_{a+1}^n, ..., x_{a+b}^n)$ donde $x_1, x_2, ..., x_a$ son variables numéricas que viven en $\mathbb{R}$ y $x_{a+1}, x_{a+2}, ..., x_{a+b}$ son variables categóricas que viven en el espacio $\mathbb{E}$ de todas las clases posibles entre todas las variables categóricas; entonces queremos construir una red neuronal artificial para predecir y comparar el comportamiento de un conjunto de observaciones $\mathfrak{X} = \{(X_v, Y_v\}_{s\in\mathfrak{v}}$ distintas a las N para el conjunto de prueba. La red tendrá los parámetros:

a) M indica la cantidad de datos sobre los N iniciales que serán tomados para el procedimiento de entrenar a la red neuronal artificial.

b) h indica unidades en la capa de en medio.

c) *maxit* indica la cantidad máxima de iteraciones que tendrá la optimización de los pesos.

d) λ indica el decaimiento de los pesos.

e) $MaxNWts$ indica la cantidad máxima de pesos que permite la red.

f) *rang* indica el máximo valor absoluto en los primeros pesos aleatorios del método.

g) *abstol* indica la tolerancia máxima para el criterio de acoplamiento $E+\lambda C(f_n)$ con la que el procedimiento de optimización puede terminar.

h) *reltol* indica que si la reducción en el criterio de acoplamiento es menor a $1 - reltol$ el procedimiento termina.

Y por último se tomarán los restantes $N - M$ de los N iniciales para la validación. Entonces el pseudocódigo que nos ayuda a ejemplificar la creación de redes neuronales artificiales con dichos parámetros es:

Algoritmo 2 Red Neuronal Artificial

Entrada:
$\mathfrak{L} = \{(X_n, Y_n) | n = 1, 2, ..., N; X_n \in \mathbb{R}^{a+b}\}$
$\mathfrak{X}$
Número de nodos en la capa de en medio h
Cantidad de elementos para el conjunto de entrenamiento M
Decaimiento de los pesos λ
Cantidad máxima de pesos $MaxNWts$
Cantidad máxima de iteraciones $maxit$
Máximo valor absoluto en los primeros pesos aleatorios $rang$

Procedimiento CREAR UNA RED NEURONAL ARTIFICIAL (Usando una muestra de M elementos de $\mathfrak{L}$)
    Transformar todos los datos hacía el intervalo $(-1, 1)$
    Construir la topología de la red
    Calcular la cantidad de pesos en la red
    Guardar dicho valor en $nwts$
    si $nwts > MaxNWts$ **entonces**
        Terminar el proceso
    si $rang = NA$ **entonces**
        $rang \leftarrow 0.5$
    Generar pesos aleatorios entre el valor absoluto $rang$
    Guardar dichos pesos en el vector wts
    Construir la función f_n
    Calcular $E + \lambda C(f_n)$
    Guardar dicho valor en E_1
    $nit \leftarrow 0$
    $redfit \leftarrow 1$
    mientras $nit < maxit$ & $redfit > 1 - reltol$
        Optimizar los pesos según el método BFGS (haciendo una iteración)
        $nit \leftarrow nit + 1$
        Calcular $E + \lambda C(f_n)$ con los nuevos pesos
        Guardar dicho valor en E_2
        $redfit \leftarrow E_1 - E_2$
        $E_1 \leftarrow E_2$
Procedimiento VALIDAR LA RED NEURONAL ARTIFICIAL (Con los datos de $\mathfrak{L}$ que no fueron usados en el entrenamiento)
    Calcular E
    si $E > 0.4$ **entonces**
        Repetir el procedimiento de crear una red con distintos parámetros o datos
Procedimiento PROBAR LA RED NEURONAL ARTIFICIAL (Con los datos de $\mathfrak{X}$)
    Calcular E
    Guardar dicho error en Err

Salida: La red creada tiene $nwts$ pesos y un error de Err

2.2.2. Análisis del algoritmo

Teorema 2.2.1. *El costo en tiempo de ejecución del algoritmo 2 tiene una razón de crecimiento de* $\Theta(N^2)$ *en el peor caso.*

Demostración: Para analizar el algoritmo empecemos considerando los procedimientos descritos uno por uno:

		Costo	*# veces*
	CREAR LA RED		
1:	Transformar datos hacía $(-1, 1)$	c_1	N
2:	Construir la topología de la red	c_2	N
3:	Calcular la cantidad de pesos en la red	c_3	N
4:	Guardar dicho valor en $nwts$	c_4	1
5:	**si** $nwts > MaxNWts$ **entonces**	c_5	1
6:	Terminar el proceso	c_6	1
7:	**si** $rang = NA$ **entonces**	c_7	1
8:	$rang \leftarrow 0.5$	c_8	1
9:	Generar pesos aleatorios	c_9	N
10:	Guardar dichos pesos en el vector wts	c_{10}	N
11:	Construir la función f_n	c_{11}	1
12:	Calcular $E + \lambda C(f_n)$	c_{12}	1
13:	Guardar dicho valor en E_1	c_{13}	1
14:	$nit \leftarrow 0$	c_{14}	1
15:	$redfit \leftarrow 1$	c_{15}	1
16:	**mientras** $nit < maxit$ & $redfit > 1 - reltol$	c_{16}	$maxit + 1$
17:	Optimizar según el método BFGS	c_{17}	$maxit * O(N^2)$
18:	$nit \leftarrow nit + 1$	c_{18}	$maxit$
19:	Calcular $E + \lambda C(f_n)$ con los nuevos pesos	c_{19}	$maxit$
20:	Guardar dicho valor en E_2	c_{20}	$maxit$
21:	$redfit \leftarrow E_1 - E_2$	c_{21}	$maxit$
22:	$E_1 \leftarrow E_2$	c_{22}	$maxit$
	VALIDAR LA RED	*Costo*	*# veces*
23:	Calcular E	c_{23}	1
24:	**si** $E > 0.4$ **entonces**	c_{24}	1
25:	Repetir el procedimiento de crear una red con distintos parámetros o datos	c_{25}	1
	PROBAR LA RED	*Costo*	*# veces*
26:	Calcular E	c_{26}	1
27:	Guardar dicho error en Err	c_{27}	1

Donde se puede ver que el costo en tiempo o la función Υ para este algoritmo

tiene la forma $A + BN + CN_2$, donde A, B, C son constantes que dependen de las cosntantes c_i, siendo así el término de mayor orden el N^2 correspondiente al $c_{17} * maxit * O(N^2)$ en el pseudocódigo. $\triangle$

2.2.3. Software

Ahora usamos el paquete ***nnet*** en *R Console*. Para crear redes neuronales se utilizó el comando *nnet*, el cual se usa de la siguiente manera:
`nnet (formula, data, subset, size, linout, rang, decay, maxit, MaxNWts)`

formula: expresión que indica la variable a predecir y las variables a utilizar. $Y \sim x_1 + x_2 + ... + x_n$.

data: tabla con las observaciones.

subset: arreglo de índices que indican los datos a utilizar de la tabla en data.

size: cantidad de nodos en la capa de en medio.

skip: indica si las entradas se conectan con las salidas.

linout: indica si se cambia a unidades de salida lineales.

rang: máximo valor absoluto de los pesos.

decay: parámetro para el decaimiento de los pesos.

maxit: cantidad máxima de iteraciones.

MaxNWts: cantidad máxima de pesos aceptados.

Para escoger el mejor modelo se varía la cantidad de unidades de en medio y se guarda un registro de su efectividad, para luego escoger la mejor. Para medir la efectividad de un modelo se hace lo siguiente, que entrega un conjunto de datos donde a cada valor a predecir se le asigna el vector de probabilidades de pertenecer a tal clase o la clase con mayor probabilidad de ser.:
`predict (nnet, newdata, type = c (“raw”, “class”))`

nnet: la red obtenida con nnet.

newdata: conjunto de validación para la red.

type: tipo de salidas entre vector de probabilidad o clases.

Para ver el error de clasificación se puede hacer de la siguiente manera, que devuelve una tabla donde se puede ver cuantos fueron clasificados correctamente y cuantos no lo fueron.:

```
table (predict, newdata [, Y])
```

predict: predicción obtenida con predict de tipo clase.

newdata: conjunto de validación de la red.

Y: variable a predecir.

Entonces se debe escoger la red con menor error de clasificación y luego se vuelve a medir el error para esa red, pero con el conjunto de prueba.

2.3. Algoritmo para máquinas de vectores

2.3.1. Pseudocódigo

Sean N observaciones (X_i, Y_i) que tienen variables de valores numéricos o categóricos, $X_i = (x_1^i, x_2^i, ..., x_a^i, x_{a+1}^i, ..., x_{a+b}^i)$ donde $x_1, x_2, ..., x_a$ son variables numéricas que viven en $\mathbb{R}$ y $x_{a+1}, x_{a+2}, ..., x_{a+b=n}$ son las representaciones numéricas de variables categóricas que viven en el espacio $\mathbb{E}$ de todas las clases posibles entre todas las variables categóricas; entonces queremos construir una máquina de vectores de soporte para predecir y comparar el comportamiento de un conjunto de observaciones $\mathfrak{X} = \{(X_v, Y_v\}_{s \in \mathfrak{v}}$ distintas a las N para el conjunto de prueba. La máquina de vectores de soporte tendrá los parámetros:

a) M la cantidad de datos del conjunto de entrenamiento

b) γ la penalización agregada al error al limitar el efecto de los vectores de soporte

c) *kernel* el tipo de kernel a utilizar entre *polinomial*, *tangente*, *gauss*, *laplace*

d) *tipo* el tipo de máquina entre γ clasificación y ν búsqueda de extraños

e) Variación del volumen de la región sin datos extraños ν

f) Parámetro para los kernel de base radial σ

g) Multiplicador de la escala en los kernel polinomial y tangente hiperbólica *escala*

h) Sesgo en los kernel polinomial y tangente hiperbólica *sesgo*

i) Exponente del kernel polinomial *grado*

j) Además se tomarán los restantes $N - M$ de los N iniciales para la validación.

Entonces el pseudocódigo propuesto como ejemplo para la creación de máquinas de vectores de soporte con dichos parámetros es:

Algoritmo 3 Máquina de vectores de soporte

Entrada:
$\mathfrak{L} = \{(X_i, Y_i) | i = 1, 2, ..., N; X_i \in \mathbb{R}^n\}$
Cantidad de datos del conjunto de entrenamiento M
Penalización agregada al error al limitar el efecto de los vectores de soporte γ
Tipo de kernel a utilizar *kernel*
Tipo de máquina *tipo*
Variación del volumen de la región sin datos extraños ν
Parámetro para los kernel de base radial σ
Multiplicador de la escala en los kernel polinomial y tangente hiperbólica *escala*
Sesgo en los kernel polinomial y tangente hiperbólica *sesgo*
Exponente del kernel polinomial *grado*

Procedimiento CREAR UNA MÁQUINA DE VECTORES DE SOPORTE (Usando una muestra de M elementos de $\mathfrak{L}$)
    si $tipo = \gamma$ **entonces**
        Tomar el parámetro γ
        Guardarlo en la variable γ^*
        si $kernel = lineal$ **entonces**
            Construir el kernel 1
        si $kernel = polinomial$ **entonces**
            Tomar los parámetros *grado*, *sesgo*, *escala*
            Guardarlos en $grado^*$, $sesgo^*$, $escala^*$
            Construir el kernel 3
        si $kernel = tangente$ **entonces**
            Tomar los parámetros *sesgo*, *escala*
            Guardarlos en $sesgo^*$, $escala^*$
            Construir el kernel 4
        si $kernel = gauss$ **entonces**
            Tomar el parámetro σ
            Guardarlo en σ^*
            Construir el kernel 2

si $kernel = laplace$ **entonces**
Tomar el parámetro σ
Guardarlo en σ^*
Construir el kernel 5
Optimizar el problema 1.28
Guardar los valores óptimos del problema en a_i^*
Con dichos valores construir w según la ecuación 1.26
Guardar dicho vector en w^*
Construir la función u según la ecuación 1.27
si $tipo = \nu$ **entonces**
Tomar el parámetro γ
Guardarlo en la variable γ^*
Tomar el parámetro ν
Guardarlo en ν^*
si $kernel = lineal$ **entonces**
Construir el kernel 1
si $kernel = polinomial$ **entonces**
Tomar los parámetros $grado$, $sesgo$, $escala$
Guardarlos en $grado^*$, $sesgo^*$, $escala^*$
Construir el kernel 3
si $kernel = tangente$ **entonces**
Tomar los parámetros $sesgo$, $escala$
Guardarlos en $sesgo^*$, $escala^*$
Construir el kernel 4
si $kernel = gauss$ **entonces**
Tomar el parámetro σ
Guardarlo en σ^*
Construir el kernel 2
si $kernel = laplace$ **entonces**
Tomar el parámetro σ
Guardarlo en σ^*
Construir el kernel 5
Optimizar el problema 1.29
Guardar el vector w en w^*
Construir la función u según la ecuación 1.27
Procedimiento VALIDAR LA MÁQUINA DE VECTORES DE SOPORTE (Con los datos de $\mathfrak{L}$ que no fueron usados en el entrenamiento)
Calcular el error de clasificación
Guardar dicho error en E
si $E > 0.4$ **entonces**
Repetir el procedimiento de crear una máquina de vectores de soporte con distintos parámetros o datos

59: **Procedimiento** PROBAR LA MÁQUINA DE VECTORES DE SOPORTE (Con los datos de $\mathfrak{X}$)
60: Calcular el error de clasificación
61: Guardar dicho error en E
62: Guardar dicho error en Err
Salida: La máquina creada tiene un error de Err

2.3.2. Análisis del algoritmo

Teorema 2.3.1. *El costo en tiempo de ejecución del algoritmo 3 tiene una razón de crecimiento de* $\Theta(N^2)$ *en el peor caso.*

Demostración: Para analizar el algoritmo empecemos considerando los procedimientos descritos uno por uno:

CREAR LA MÁQUINA	*Costo*	# *veces*
1: **si** $tipo = \gamma$ **entonces**	c_1	1
2: Tomar el parámetro γ	c_2	1
3: Guardarlo en la variable γ^*	c_3	1
4: **si** $kernel = lineal$ **entonces**	c_4	1
5: Construir el kernel 1	c_5	1
6: **si** $kernel = polinomial$ **entonces**	c_6	1
7: Tomar los parámetros *grado*, *sesgo*, *escala*	c_7	1
8: Guardarlos en $grado^*$, $sesgo^*$, $escala^*$	c_8	1
9: Construir el kernel 3	c_9	1
10: **si** $kernel = tangente$ **entonces**	c_{10}	1
11: Tomar los parámetros *sesgo*, *escala*	c_{11}	1
12: Guardarlos en $sesgo^*$, $escala^*$	c_{12}	1
13: Construir el kernel 4	c_{13}	1
14: **si** $kernel = gauss$ **entonces**	c_{14}	1
15: Tomar el parámetro σ	c_{15}	1
16: Guardarlo en σ^*	c_{16}	1
17: Construir el kernel 2	c_{17}	1
18: **si** $kernel = laplace$ **entonces**	c_{18}	1
19: Tomar el parámetro σ	c_{19}	1
20: Guardarlo en σ^*	c_{20}	1
21: Construir el kernel 5	c_{21}	1
22: Optimizar el problema 1.28	c_{22}	1
23: Valores óptimos del problema a_i^*	c_{23}	1

Crear la máquina *continuación*	*Costo*	# *veces*
24: Construir w según la ecuación 1.26	c_{24}	1
25: Guardar dicho vector en w^*	c_{25}	1
26: Construir la función u según la ecuación 1.27	c_{26}	1
27: **si** $tipo = \nu$ **entonces**	c_{27}	1
28: Tomar el parámetro γ	c_{28}	1
29: Guardarlo en la variable γ^*	c_{29}	1
30: Tomar el parámetro ν	c_{30}	1
31: Guardarlo en ν^*	c_{31}	1
32: **si** $kernel = lineal$ **entonces**	c_{32}	1
33: Construir el kernel 1	c_{33}	1
34: **si** $kernel = polinomial$ **entonces**	c_{34}	1
35: Tomar los parámetros *grado*, *sesgo*, *escala*	c_{35}	1
36: Guardarlos en $grado^*$, $sesgo^*$, $escala^*$	c_{36}	1
37: Construir el kernel 3	c_{37}	1
38: **si** $kernel = tangente$ **entonces**	c_{38}	1
39: Tomar los parámetros *sesgo*, *escala*	c_{39}	1
40: Guardarlos en $sesgo^*$, $escala^*$	c_{40}	1
41: Construir el kernel 4	c_{41}	1
42: **si** $kernel = gauss$ **entonces**	c_{42}	1
43: Tomar el parámetro σ	c_{43}	1
44: Guardarlo en σ^*	c_{44}	1
45: Construir el kernel 2	c_{45}	1
46: **si** $kernel = laplace$ **entonces**	c_{46}	1
47: Tomar el parámetro σ	c_{47}	1
48: Guardarlo en σ^*	c_{48}	1
49: Construir el kernel 5	c_{49}	1
50: Optimizar el problema 1.29	c_{50}	$O(N^2)$
51: Guardar el vector w en w^*	c_{51}	1
52: Construir la función u según la ecuación 1.27	c_{52}	1
Validar la máquina		
53: Calcular el error de clasificación	c_{53}	1
54: Guardar dicho error en E	c_{54}	1
55: **si** $E > 0.4$ **entonces**	c_{55}	1
56: Repetir el procedimiento de crear una máquina de vectores de soporte con distintos parámetros o datos	c_{56}	$--$
Probar la máquina		
57: Calcular el error de clasificación	c_{57}	1
58: Guardar dicho error en E	c_{58}	1
59: Guardar dicho error en Err	c_{59}	1

Así el costo en tiempo del algoritmo tiene la forma de $A + BN + CN^2$, donde A, B, C, D, E, F son constantes que dependen de las cosntantes c_i, y el término de mayor orden es el N^2 que corresponde a $c_{50} * O(N^2)$ en el código. $\triangle$

2.3.3. Software

Para las máquinas de vectores de soporte usamos el paquete ***e1071*** en *R Console* con el comando ,*svm*, el cual se usa de la siguiente manera:
`svm (formula, data subset, scale, type, kernel, degree, gamma, coef0, cost, nu, tolerance, linout, rang, decay, maxit, MaxNWts)`

formula: expresión que indica la variable a predecir y las variables a utilizar. $Y \sim x_1 + x_2 + ... + x_n$.

data: tabla con las observaciones.

subset: arreglo de índices que indican los datos a utilizar de la tabla en data.

scale: indica si se debe normalizar los datos.

type: tipo de máquina de vectores de soporte entre C-classification, one-classification.

kernel: tipo de kernel a escoger entre linear, polynomial, radial basis, sigmoid.

degree: grado del kernel polinomial.

gamma: escala de los kernel.

coef0: sesgo de los kernel.

cost: variable que penaliza.

nu: parámetro de la búsqueda de extraños.

tolerance: tolerancia para criterio de parada.

Para escoger el mejor modelo se varían los parámetros y se guarda un registro de su efectividad, para luego escoger la mejor. Para medir la efectividad de un modelo se hace lo siguiente, que obtiene un conjunto de datos donde a cada valor a predecir se le asigna el vector de probabilidades de pertenecer a tal clase o la clase con mayor probabilidad de ser:
`predict (svm, newdata, type = c ( "class"))`

svm: máquina de vectores de soporte obtenida con svm.

newdata: conjunto de validación para la svm.

type: para que el tipo de salidas sean clases.

Para ver el error de clasificación se puede hacer de la siguiente manera, que devuelve una tabla donde se puede ver cuantos fueron clasificados correctamente y cuantos no lo fueron:

```
table (predict, newdata [, Y])
```

predict: predicción obtenida con predict de tipo clase.

newdata: conjunto de validación de la svm.

Y: variable a predecir.

Entonces se escoge la máquina con menor error de clasificación y luego se vuelve a medir el error para esa máquina, pero con el conjunto de prueba.

3. DESCRIPCIÓN DE LOS DATOS Y METODOLOGÍA

3.1. Datos

Como se ha dicho anteriormente los algoritmos necesitan datos de entrenamiento, de validación y de prueba, la utilidad de realizar los tres conjuntos es:

Primero: El algoritmo necesita aprender a clasificar con datos reales así se le da el conjunto de entrenamiento para que aprenda de ellos y pueda crear el modelo o reglas a evaluar para cada dato y lo clasifique según ellas.

Segundo: Luego de ser construido el modelo puede que este tenga ciertos parámetros con la posibilidad de ser modificados, así éstos se modifican para escoger el mejor rendimiento del algoritmo pero no se hace con los mismos datos donde la efectividad ya estaría garantizada, así se necesita un segundo grupo de datos para corroborar la validez del modelo para ser generalizado y no se limite a los datos de entrenamiento y ahí es donde surge el conjunto de validación para ayudarnos a aceptar un modelo.

Tercero: Una vez se ha aprobado y escogido el modelo con los mejores resultados en la validación se procede a medir la eficiencia del modelo pero escogerlo con los datos de entrenamiento o de validación no da mayor información ya que está construido para tener buen resultado en ellos, de donde se necesita un tercer conjunto para medir la efectividad del modelo y éste es el conjunto de prueba.

El problema de categorizar el riesgo en créditos es encontrar la etiqueta del cliente entre las categorías de riesgo A, B, C, D, E, F, donde A es la más riesgosa y F la de menor riesgo. En la tabla siguiente se tienen las proporciones de las categorías en la población a clasificar:

Tabla 3.1. Tabla de las proporciones según las clases A, B, C, D, E, F en la población a clasificar

Clase	A	B	C	D	E	F
%	2.43	4.64	5.93	19.10	24.92	42.98

La base de datos fue construida de la información guardada en el año 2013 para predecir el año 2014, dicha información está contenida en 84 variables, 9 representadas por números de carácter continuo, 7 tipo banderas en representación de 3 variables categóricas y 68 numéricas de carácter discreto. La manera en que se realizó la separación en los conjuntos de entrenamiento, validación y prueba fue tomando el 64 % para entrenamiento, el 16 % para validación y el 20 % para prueba considerando aproximaciones. En los conjuntos de entrenamiento, validación y prueba se consideró tomar la misma proporción de cada clase que del conjunto a analizar.

3.2. Metodología

3.2.1. Para los árboles de clasificación

Para realizar la categorización utilizando el método de árboles de clasificación no se necesita manipular los datos puesto que analiza cada corte según los valores existentes y relaciones de orden, adicionalmente el único parámetro que se cambiará es el parámetro de la complejidad que dice el factor por el cual debe ser capaz de disminuir el error de clasificación cuando se realiza un corte, la forma de variarlo será desde el valor de 0.0001 incrementando 0.0001 en cada iteración hasta el valor de 0.01.

Para la comparación de la eficiencia de cada una de las combinaciones de parámetros, que en el caso de los árboles de clasificación solo se tiene un parámetro, se consideran cuatro mediciones las cuales son las siguientes:

Tabla 3.2. Tabla con las medidas de clasificación

Medida	**Símbolo**	**Comentario**
True perfect rate	tp	Porcentaje de los clientes predichos en su clase
True class rate	tc	Porcentaje de los clientes predichos en su clase o una clase adyacente

Medida	Símbolo	Comentario
False positive rate	fp	Porcentaje de los clientes predichos en una clase más alta a las adyacentes de la verdadera clase
False negative rate	fn	Porcentaje de los clientes predichos en una clase más baja a las adyacentes de la verdadera clase

La razón por la que se consideraron dichas medidas es porque el hecho de que el algoritmo encuentre a una clase muy cercana a la clase verdadera es algo en sí es algo bueno porque al tener seis clases, al menos se tiene que el cliente se le considera como de una categoría de clases con mayor probabilidad que la de pertenecer a la clase que se predicho y ayuda a convencerse de que el método si aporta información.

3.2.2. Para las redes neuronales artificiales

Para realizar la clasificación utilizando el método de redes neuronales artificiales se necesita manipular los datos, ya que que analiza cada valor en una función de transferencia que funciona de manera más efectiva en el intervalo $(-1, 1)$, adicionalmente el único parámetro que se cambiará es el de la cantidad de nodos en la capa de en medio, la forma de variarlo será desde el valor de 1 a 50 moviéndose de 1 en 1, aunque lo malo de considerar dichos valores es que no todos los valores crean una predicción efectiva ya que a veces encuentran la comodidad de predecir todo a la clase más común entonces no todos los valores tendrán medidas significativas. Las medidas a utilizar son las mismas que para los árboles de clasificación.

3.2.3. Para las máquinas de vectores de soporte

Para realizar la clasificación utilizando el método de máquinas de vectores de soporte se necesita manipular los datos puesto que analiza cada valor de la misma manera y se requiere de distancias que representen lo mismo de variable a variable, adicionalmente los parámetros que se cambiarán son los parámetros de costo y gamma que son valores en la función a optimizar, el coste significa el peso que se le da al error de clasificación y gamma el parámetro del mapeo implícito a realizar; donde la forma de variarlos es para el coste en las potencias de 10 desde la potencia -1 a la 2 y para gamma a través de los valores de 0.1, 0.4, 0.7 y 1, profundizando al segundo decimal en los casos de la mejor eficiencia. Adicionalmente y dado a la naturaleza de los parámetros se puede realizar una segunda corrida con otros valores de los parámetros en el intervalo en el cual según la primera corrida se considere

está la máxima efectividad. Las medidas a utilizar son las mismas que para los casos de árboles de clasificación y redes neuronales artificiales.

4. RESULTADOS

4.1. Árbol de decisión

Al aplicar el algoritmo de árboles de decisión en los conjuntos de datos para entrenamiento, validación y prueba descritos, se obtuvieron los siguientes resultados sobre el conjunto de validación:

Tabla 4.1. Tabla con las medidas de la clasificación con los árboles de clasificación en el conjunto de validación. Fuente: Obtenida de los resultados con R.

cp	**tp**	**tc**	**fp**	**fn**
0.0001	0.5531	0.8678	0.0371	0.0950
0.001	0.5471	0.8757	0.0362	0.0881
0.002	0.5460	0.8757	0.0346	0.0889
0.003	0.5379	0.8601	0.0345	0.1055
0.004	0.5379	0.8601	0.0345	0.1055
0.005	0.5371	0.8519	0.0459	0.1021
0.006	0.5371	0.8519	0.0459	0.1021
0.007	0.5371	0.8519	0.0459	0.1021
0.008	0.5371	0.8519	0.0459	0.1021
0.009	0.5165	0.8211	0.0546	0.1243
0.010	0.5165	0.8211	0.0546	0.1243

Donde se observa que el árbol que tiene más efectividad en la clasificación, es decir con la tp más alta es el árbol que tiene el parámetro de complejidad de 0.0001. Y con esta combinación de parámetros tenemos los siguientes resultados en la predicción sobre el conjunto de prueba(en porcentajes), donde se tienen medidas de clasificación de $tp = 0.5460$, $tc = 0.8635$, $fp = 0.0378$, $fn = 0.0987$:

Tabla 4.2. Tabla con la distribución de la predicción (horizontal) del mejor árbol de clasificación en el conjunto de prueba. Fuente: Obtenida de los resultados con R.

Clase	**1**	**2**	**3**	**4**	**5**	**6**
1	0.5168	0.2815	0.0882	0.1765	0.1933	0.0588
2	0.3319	0.5168	0.2185	0.3235	0.1386	0.0672
3	0.3067	0.6176	2.1301	1.1218	0.4243	0.1470
4	0.7058	1.9284	2.7519	9.6420	5.0290	2.1637
5	0.2689	0.5462	0.4033	3.5249	5.4197	4.1089
6	0.3025	0.7520	0.3445	4.3064	13.7089	36.4339

4.2. Redes neuronales

Al aplicar el algoritmo de redes neuronales en los conjuntos de datos para entrenamiento, validación y prueba descritos, se obtuvieron los siguientes resultados sobre el conjunto de validación, tomando en consideración que se realizaron 7949 corridas para verificar si una red con una unidad en la capa intermedia era capaz de generar una predicción para las seis clases, pero unas clases eran las predominantes, por lo que la clasificación que efectua una red con esa topología no es considerada para más análisis; análogamente se realizaron corrridas para una red con 4 unidades con el mismo resultado, de donde se concluye que lo mismo sucedería para redes con 2 y 3 unidades en la capa intermedia:

Tabla 4.3. Tabla con las medidas de la clasificación con las redes neuronales en el conjunto de validación. Fuente: Obtenida de los resultados con R.

# nodos	**tp**	**tc**	**fp**	**fn**
5	0.5458	0.8657	0.0335	0.1008
10	0.5277	0.8580	0.0239	0.1181
15	0.5457	0.8664	0.0383	0.0953
20	0.5429	0.8612	0.0456	0.0931
25	0.5317	0.8600	0.0249	0.1151
29	0.5494	0.8706	0.0338	0.0956
30	0.5310	0.8602	0.0252	0.1145
35	0.5489	0.8708	0.0355	0.0937
40	0.5481	0.8718	0.0320	0.0962
45	0.5466	0.8702	0.0377	0.0921
50	0.5483	0.8675	0.0349	0.0976

Donde se observa que la red que tiene más efectividad en la clasificación es la red que tiene 29 unidades en la capa intermedia. Y con esta combinación de parámetros tenemos los siguientes resultados en la predicción sobre el conjunto de prueba (en porcentajes), donde se tienen medidas de clasificación de $tp = 0.5460$, $tc = 0.8672$, $fp = 0.0324$, $fn = 0.1004$:

Tabla 4.4. Tabla con la distribución de la predicción (horizontal) de la mejor red en el conjunto de prueba. Fuente: Obtenida de los resultados con R.

Clase	**1**	**2**	**3**	**4**	**5**	**6**
1	0.5713	0.4075	0.1176	0.1471	0.1765	0.1092
2	0.3487	0.4327	0.2101	0.2899	0.1765	0.0798
3	0.2563	0.4159	1.8234	0.8529	0.2941	0.0756
4	0.7058	2.0502	3.0964	9.6589	4.6887	1.7772
5	0.2353	0.6302	0.3949	3.6930	4.9744	3.8022
6	0.3151	0.7058	0.2941	4.4534	14.6038	37.1355

4.3. Máquinas de vectores de soporte

Al aplicar el algoritmo de redes neuronales en los conjuntos de datos para entrenamiento, validación y prueba descritos, se obtuvieron los siguientes resultados sobre el conjunto de validación:

Tabla 4.5. Tabla con las medidas de la clasificación con las máquinas de soporte en el conjunto de validación. Fuente: Obtenida de los resultados con R.

Costo	**Gamma**	**tp**	**tc**	**fp**	**fn**
0.1	0.1	0.3114	0.5079	0.4573	0.0349
0.1	0.4	0.0961	0.1884	0.8037	0.0078
0.1	0.7	0.1926	0.2817	0.7103	0.0080
0.1	1	0.1873	0.2710	0.7220	0.0070
1	0.1	0.4987	0.8183	0.1064	0.0752
1	0.4	0.2906	0.6026	0.3222	0.0752
1	0.7	0.2643	0.5760	0.3504	0.0736
1	1	0.2546	0.5639	0.3625	0.0735
10	0.1	0.5458	0.8508	0.0505	0.0987
10	0.4	0.5396	0.8066	0.0228	0.1706
10	0.7	0.5379	0.7999	0.0197	0.1804

10	1	0.5393	0.7987	0.0182	0.1831
100	0.1	0.5436	0.8484	0.0493	0.1023
100	0.4	0.5378	0.8050	0.0232	0.1717
100	0.7	0.5375	0.7981	0.0211	0.1808
100	1	0.5365	0.7960	0.0206	0.1834

Donde se observa que los mejores resultados obtenidos son los que da la máquina con costo 10 y gamma 0.1, además se puede observar que las medidas de la clasificación *tp* son inversamente proporcionales a gamma para cualquiera de los costos, de donde se asume que dicha conducta se cumpliría incluso para los valores de gamma menores a 0.1, por lo tanto como en el caso de las máquinas de vectores de soporte los parámetros son continuos podemos realizar otra corrida con nuevos parámetros, en este caso para gammas menores a 0.1 con el costo 10. Los nuevos parámetros a considerar son los siguientes gammas: 0.001, 0.004, 0.007, 0.01, 0.04, 0.07.

Tabla 4.6. Tabla con las medidas de la clasificación con las máquinas de soporte para la segunda corrida en el conjunto de validación. Fuente: Obtenida de los resultados con R.

Costo	**Gamma**	**tp**	**tc**	**fp**	**fn**
10	0.001	0.5118	0.8757	0.0601	0.0641
10	0.004	0.5160	0.8671	0.0706	0.0622
10	0.007	0.5211	0.8612	0.0773	0.0614
10	0.010	0.5199	0.8556	0.0836	0.0609
10	0.040	0.5250	0.8497	0.0880	0.0623
10	0.070	0.5198	0.8410	0.0871	0.0719

Donde se observa que la máquina que tiene más efectividad es la máquina que tiene un valor de gamma de 0.001 y el costo de 10 ya que aunque no tiene el mejor valor efectividad de la clasificación, al comparar lo sucedido en la medida de clasificación por clases cercanas o *tc* se observa que este valor es menor en las máquinas con mayor valor en *tp*. Y con esta combinación de parámetros tenemos los siguientes resultados en la predicción sobre el conjunto de prueba(en porcentajes), donde se tienen medidas de clasificación de $tp = 0.5460$, $tc = 0.8672$, $fp = 0.0324$, $fn = 0.1004$:

Tabla 4.7. Tabla con la distribución de la predicción (horizontal) de la mejor máquina en el conjunto de prueba. Fuente: Obtenida de los resultados con R.

Clase	**1**	**2**	**3**	**4**	**5**	**6**
1	0.9831	0.9957	0.3445	0.6680	0.5000	0.3403
2	0.1176	0.2605	0.1050	0.1933	0.1975	0.0882
3	0.5168	1.2184	3.6930	4.2433	1.4621	0.4705
4	0.3571	0.9537	1.1092	6.3692	3.8358	1.7646
5	0.2689	0.7688	0.5504	5.2643	9.5328	9.9992
6	0.1891	0.4453	0.1344	2.3569	9.3858	30.3168

CONCLUSIONES

1. Los tres algoritmos presentados, junto a ciertos supuestos en cuanto a la información con la que operan, muestran no ser ineficientes para la predicción de una situacón crediticia en Guatemala, si se toma como positivo el predecir un estado de la situación o un estado similar.

2. El periodo de análisis de un año muestra resultados positivos, que podrían ser optimizados con el manejo de las variables usadas en los algoritmos.

3. Los árboles de clasificación llegaron a predecir de manera correcta en un 54.65 % de los casos y de manera eficiente para un 86.35 %, las redes neuronales lograron predecir de manera correca en un 54.59 % de los casos y de manera eficiente en un 86.71 %, mientras que las máquinas de vectores de soporte tuvieron una precisión del 51.15 % junto a una eficiencia de 87.42 % .

4. Los tres métodos obtuvieron resultados positivos y sus eficiencias varían en pocos décimos, pero, aunque los árboles y las redes tengan más precisión en la clasificación, las máquinas tienen la característica de que predicen de mejor manera para el caso en que se tome la decisión de aceptar que la predicción sea en el estado o en una estado similar.

RECOMENDACIONES

1. Si se desea mejorar la predicción de la situación de crédito se debe de realizar un estudio más profundo al producto sobre el cual se hace el análisis, no se puede pretender predecir una situación de la cual no se tiene mucha información.

2. Para realizar una predicción con estos clasificadores es de mucha ayuda que previo al análisis se tenga la información debidamente recolectada, organizada y guardada en un sistema de bases de datos.

3. El interesado en realizar un análisis predictivo conel software estadístico R y los comandos en este trabajo presentados, debe considerar leer la documentación propuesta por el autor de los paquetes a utilizar ya que los resultados varían con la variación de los parámetros a utilizar.

BIBLIOGRAFÍA

[1] M. Berry y G. Linoff. *Data Mining Techniques.* 2.ª ed. Wiley Publishing, Inc. Indiana,2004.

[2] R. Timofeev. *Classification and Regression Trees (CART) Theory and Applications.* Tesis para aplicar al título de maestro en arte. Berlín: Humboldt University, Center of Applied Statistics and Economics, 2004.

[3] A. Kraus. *Recent Methods from Statistics and Machine Learning for Credit Scoring.* Tesis para aplicar al título de licenciatura. Munich: Ludwig-Maximilians-Universität, Fakultät für Mathematik, Informatik und Statistik, 2014.

[4] S. Ruiz. *Random Forest para detección de fraude en medios de pago.* Tesis para aplicar el título de maestro. Madrid Universidad Autonoma de Madrid, Escuela Politécnica Superior, Departamento de Ingeniería Informática, 2014.

[5] M. Nielsen. Neural Networks and Deep Learning. Consultado en febrero de 2016 en `http://neuralnetworksanddeeplearning.com/`.

[6] B. Ripley y W. Venables. *Modern Applied Statistic with S.* 4.ª ed. s.e., 2002.

[7] E. Kreyszig. *Introductory Functional Analysis with Applications.* John Wiley & Sons. Estados Unidos, 1978.

[8] R. Sundaram. *A First Course in Optimization Theory.* Cambridge University Press. Estados Unidos,1996.

[9] J. Platt. *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines.* Technical Report MSR-TR-98-14. 21 de abril de 1998.

[10] A. Karatzoglou, D. Meyer, H. Hornik *Support Vector Machines in R. Journal of Statistical Software.* Abril de 2006. http://www.jstatsoft.org/

[11] T. Cormen, C. Leiserson, R. Rivest y C. Stein *Introduction to Algorithms.* 3.ª ed. MIT Press, EUA, 2009.

[12] O. Romero *Convergencia Global del método BFGS modificado para la minimización no-convexa irrestricta.* Tesis para aplicar al título de licenciatura. Barquisimetro: Universidad Centroccidental Lisandro Alvarado, Decanato de Ciencias y Tecnología, Licenciatura en Ciencias Matemáticas, 2012.

A. Método de Optimización BFGS

Si se busca la solución del problema de optimización:

$$\min_x f \quad s.a. \quad x \in \mathbb{R}^n. \tag{A.1}$$

Con el método cuasi-Newton, que como el método de Newton se basan en el desarrollo de un modelo cuadrático de la función objetico construido con la información de la curvatura de la función, solo que mientras Newton busca la inforamción de la curvatura directamente de la hessiana (matriz de segundas derivadas parciales) los cuasi-Newton obtienen la información por medio del comportamiento de la función y el descenso en el gradiente de manera iterativa. Por lo que los métodos cuasi-Newton son aveces más eficientes para este tipo de problemas que el de Newton ya que las segundas derivadas no son necesarias. Entonces el análisis es el siguiente:

El método busca minimizar a f suponiendo primero que se está en un punto de aproximación inicial y en base a ese punto tomar pasos sobre la función para acercarse al mínimo por lo que se quiere realizar pequeños cambios en nuestra posición de tal forma que generen pequeños cambios negativos a la función hasta llegar al «equilibrio», así si nos movemos en la dirección v, dichos cambios los podemos ver de la siguiente manera:

$$\Delta f = \nabla f \cdot \Delta v. \tag{A.2}$$

Así, sea x_k el valor de la aproximación en la iteración k la cual se construye por medio de la anterior al agregarle un paso, entonces tenemos a x_0 vector para iniciar el método y de él se construye x_k para $k \geq 1$ como sigue:

$$x_k := x_{k-1} + a_{k-1} d_{k-1}.$$

Donde a_0 es un escalar que indica la longitud del paso a dar y d_0 que indica el cambio en la dirección. Para garantizar el signo negativo en la ecuación A.2 las direcciones d se definen como sigue:

$$d := -\mathcal{B}^{-1}\nabla f(x). \tag{A.3}$$

Con $\mathcal{B}$ matriz definida positiva, lo que implica que:

$$\Delta f = -\nabla f(x)^T \mathcal{B}^{-1}\nabla f(x) < 0.$$

Entonces esas direcciones así definidas con $\mathcal{B}$ una aproximación a la matriz hessiana que en cada iteración se actualiza, ayudan a que se dé el descenso en el gradiente, entonces solo nos falta definir las posibles longitudes de salto a que ayuden a dar una disminución considerable, para lo cual se tiene la siguiente condición sobre a_k en la iteración k:

$$f(x_k + a_k d_k) < f(x_k). \tag{A.4}$$

Pero esta condición no es suficiente ya que puede no disminuir lo suficiente, entonces se le pide la llamada condición de Armijo [12, p. 6] a la disminución que sea además de negativa que sea proporcional a la longitud de paso y a la derivada direccional en la dirección d_k es decir se le da más fuerza a la condición

$$f(x_k + a_k d_k) - f(x_k) \leq c_1 a_k \nabla f_k \cdot d_k. \tag{A.5}$$

Donde c_1 es una constante pequeña, generalmente se escoge 10^{-4} [12, p. 6]. Ahora con esta condición se considerarán solo valores pequeños, pero puede que los valores que llegue a tomar sean demasiado pequeños y por ende el algoritmo no haga un progreso considerable, así se da una condición extra para evitar esto, la cual es la siguiente:

$$\nabla f(x_k + a_k d_k) \cdot d_k \geq c_2 \nabla f(x_k) \cdot d_k. \tag{A.6}$$

Donde $c_2 \in (c_1, 1)$ y la condición indica que la curvatura generada por a_k es mayor que c_2 veces la curvatura anterior. Así para la optimización construimos el modelo cuadrático de nuestar función objetivo:

$$\varphi_k(d_k) := f(x_k) + \nabla f(x_k) \cdot d_k + \frac{1}{2} d_k^T \mathcal{B}_k d_k. \tag{A.7}$$

Donde el valor de esta función y su gradiente evaluados en $d_k = 0$ son $f(x_k)$ y $\nabla f(x_k)$ respectivamente. Por lo tanto como la dirección de acercamiento está dada en términos de $\mathcal{B}$ solo falta definir como se construyen las matrices definidas positivas que busan acercarse a la matrix Hessiana, lo que se consigue al actualizarla

de manera sencilla al tener en cuenta la curvatura medida en el punto anterior, así si ya se generó el vector x_k de manera iterativa este a su vez genera su modelo cuadrático y en este modelo, puesto que lo que se hace es mejorarlo se le impone la condición de que el gradiente del modelo φ_{k+1} en los puntos actual ($d_{k+1} = 0$) y anterior ($d_{k+1} = -a_k d_k$) sea ∇f en esos puntos. En el punto actual se da la condición por la definición mientras que para el anterior queda:

$$\nabla\varphi_{k+1}(-a_k d_k) = \nabla f(x_{k+1}) - a_k \mathcal{B} d_k = \nabla f(x_k).$$

De donde se tiene que:

$$a_k \mathcal{B} d_k = \nabla f(x_{k+1}) - \nabla f(x_k). \tag{A.8}$$

Donde si se cambia la notación y se considera el paso s_k como el cambio de x_k a x_{k+1} y a y_k como el cambio en $\nabla f(x_k)$ a $\nabla f(x_{k+1})$ se tiene que la ecuación A.8 queda como:

$$\mathcal{B} s_k = y_k. \tag{A.9}$$

Que es la ecuación del método de la secante para optimización numérica multivariable sin restricciones, donde la matriz $\mathcal{B}$ se necesita sea definida positiva por lo cual al multiplicar la ecuación A.9 «internamente» por s_k (desde el lado izquierdo), lo cual genera la necesidad del siguiente resultado:

$$s_k \cdot y_k > 0. \tag{A.10}$$

Dicha necesidad se cumple debido a las restricciones previamente dispuestas ya que de la ecuación A.6 con la nueva notación obtenemos que:

$$\nabla f(x_{k+1}) \cdot s_k \geq c_2 \nabla f(x_k) \cdot s_k.$$

Y sustituyendo aquí la ecuación A.8 multiplicado por s_k se tiene que:

$$s_k \cdot y_k + \nabla f(x_k) \cdot s_k \geq c_2 \nabla f(x_k) \cdot s_k.$$

Al reordenarlo se tiene lo siguiente:

$$s_k \cdot y_k \geq (c_2 - 1) \nabla f(x_k) \cdot s_k. \tag{A.11}$$

Donde como $c_2 \in (c_1, 1)$ y s_k es una dirección de descenso entonces $(c_2 - 1)\nabla f(x_k) \cdot s_k \geq 0$ quedando así la ecuación A.10 como verdadera.

Así al adicionar que se cumpla la ecuación A.6 entonces la ecuación A.9 tiene múltiples soluciones ya que la matriz genera más incognitas que las ecuaciones en la igualdad junto con las inecuaciones en el requisito de ser definida positiva. Por lo que entre las múltiples soluciones se escoge una particular, la que se aleja menos de la anterior.

Con todo lo anteriormente expuesto se llega al siguiente problema de optimización:

$$\begin{aligned} &\min_{B} \|\mathcal{B} - \mathcal{B}_k\| \\ s.a.\ \ &\mathcal{B} = \mathfrak{B}^T, \quad \mathcal{B}s_k = y_k, \quad s_k \cdot y_k > 0. \end{aligned} \tag{A.12}$$

Si reescribimos el problema donde $\mathcal{H} := \mathcal{B}^{-1}$ entonce:

$$\begin{aligned} &\min_{H} \|\mathcal{H} - \mathcal{H}_k\| \\ s.a.\ \ &\mathcal{H} = \mathcal{H}^T, \quad \mathcal{H}y_k = s_k, \quad s_k \cdot y_k > 0. \end{aligned} \tag{A.13}$$

Donde la norma utilizada para el problema de optimización es la norma de Frobenius ponderada, $\|\cdot\|$.

$$\|A\|_W := \|W^{1/2} A W^{1/2}\|_F. \tag{A.14}$$

Donde $\|\cdot\|_F$ está dada por:

$$\|A\|_F := \sum_{i=1}^{n}\sum_{j=1}^{n} a_{i\ j}^2. \tag{A.15}$$

Y $W^{1/2}$ para una matriz simétrica W que por ello se puede expresar como $\sum_{i=1}^{n} \lambda_i x_i x_i^T$ para los vectores propios x_i con sus valores propios λ_i. Entonces

$$W^{1/2} := \sum_{i=1}^{n} \lambda_i^{1/2} x_i x_i^T. \tag{A.16}$$

Y la matriz de ponderación W es cualquier matriz que cumple la condición de $Wy_k = s_k$. Pero para este problema la matriz que se utiliza la hessiana promedio definida por:

$$\int_0^1 \nabla^2 f(x_k + \tau s_k) d\tau. \tag{A.17}$$

Entonces la solución del problema es:

$$\mathcal{H}_{k+1} = (I - \rho_k y_k s_k^T)\mathcal{H}_k(I - \rho_k y_k s_k^T) + \rho_k y_k y_k^T. \tag{A.18}$$

Por lo tanto el algoritmo para éste método de optimización es:

Algoritmo 4 Algoritmo BFGS

Entrada: Función a optimizar $f(x)$
Punto de partida de la optimización x_0
Tolerancia de la convergencia ϵ
Aproximación de la Hessiana inversa $\mathcal{H}_0$

Optimización GFBS	*Costo*	*# veces*
1: $k \leftarrow 0$	c_1	1
2: **mientras** $\|\nabla f(x_k)\| > \epsilon$	c_2	$maxit + 1$
3: $d_k \leftarrow H_k \nabla f(x_k)$	c_3	$maxit$
4: Calcular a_k con las restricciones A.5 y A.6	c_4	$maxit * 3$
5: $x_{k+1} \leftarrow x_k + a_k d_k$	c_4	$maxit$
6: $s_k \leftarrow x_{k+1} - x_k$	c_4	$maxit$
7: $y_k \leftarrow \nabla f(x_{k+1}) - \nabla f(x_k)$	c_4	$maxit$
8: Calcular $\mathcal{H}_{k+1}$ según A.18	c_4	$maxit * O(n^2)$
9: $k \leftarrow k + 1$	c_4	$maxit$

B. Teorema de Kuhn-Tucker

Sean $f\colon \mathbb{R}^n \to \mathbb{R}$ y $h_i\colon \mathbb{R}^n \to \mathbb{R}$, $i = 1, 2, ..., l$ funciones diferenciables y continuas. Supongamos que x^* es un máximo local de f en el conjunto $D = U \cap \{x \in \mathbb{R}^n | h_i(x) \geq 0, i = 1, 2, ..., l\}$ donde U es un conjutno abierto en $\mathbb{R}^n$. Sea $E \subset 1, 2, ..., l$ un conjunto de restricciones efectivas en x^* y $h_E := (h_i)_{i \in E}$. Supongamos que $\rho(Dh_E(x^*)) = |E|$. Entonces se tiene que existe un vector $\lambda^* = (\lambda_1^*, \lambda_2^*, ..., \lambda_l^*) \in \mathbb{R}^l$ tal que las siguientes condiciones se cumplen:

[KT-1] $\lambda_i^* \geq 0$ y $\lambda_i^* h(x_*) = 0$ para $i = 1, 2, ..., l$.

[KT-2] $$D(f(x^*) + \sum_{i=1}^{l} \lambda_i^* Dh_i(x^*) = 0$$

La demostración del teorema puese ser encontrada en el libro de Sundaram [8] capítulo 6 sección 5.