



Uleam
UNIVERSIDAD LAICA
ELOY ALFARO DE MANABÍ



Universidad Laica Eloy Alfaro De Manabí
Facultad De Ciencias Informáticas
Tecnología De La Información

NOMBRE:

VILLAMAR PILOSO DAYANA LISSETH

TEMA:

SUPPORT VECTOR MACHINES

MATERIA:

MINERÍA DE DATOS

PROFESOR:

ING. FABRICIO RIVADENEIRA

CURSO:

SEXTO “B”

Índice

Índice	2
Introducción	3
Desarrollo teórico.....	4
Codificación y ejecución del script	6
Conclusiones.....	11
Referencias.....	12

Introducción

Las máquinas de vector soporte o Support Vector Machines (SVM) son otro tipo de algoritmo de Machine Learning supervisado aplicable a problemas de regresión y clasificación, aunque se usa más comúnmente como modelo de clasificación. Las máquinas de vector soporte suponen una generalización de un clasificador simple denominado máxima margin classifier. Sin embargo, este clasificador no puede aplicarse a sets de datos donde las clases de la variable respuesta no son separables mediante un límite lineal. Una extensión del mismo, el support vector classifier es aplicable en un mayor rango de casos. El support vector machine supone una extensión más del support vector classifier para casos con límites no lineales entre clases (clasificación binaria o de más clases).

Desarrollo teórico

Son un conjunto de técnicas estadísticas que nos permiten clasificar una población en función de la partición en subespacios de múltiples variables. Parte de la idea de dividir de forma lineal un conjunto de múltiples dimensiones. Creamos muchos hiperplanos que nos dividen las observaciones. (Baesens, 2003)

Support Vector Machine (SVM) es un algoritmo de aprendizaje automático supervisado capaz de realizar clasificación, regresión e incluso detección de valores atípicos. El clasificador lineal SVM funciona trazando una línea recta entre dos clases. Todos los puntos de datos que caen en un lado de la línea se etiquetarán como una clase y todos los puntos que caen en el otro lado se etiquetarán como el segundo. Suena bastante simple, pero hay una cantidad infinita de líneas para elegir. ¿Cómo sabemos qué línea hará el mejor trabajo para clasificar los datos? Aquí es donde entra en juego el algoritmo LSVM. El algoritmo LSVM seleccionará una línea que no solo separe las dos clases, sino que se mantenga lo más alejada posible de las muestras más cercanas. De hecho, el "vector de soporte" en "máquina de vectores de soporte" se refiere a dos vectores de posición dibujados desde el origen hasta los puntos que dictan el límite de decisión.

Las ventajas de las máquinas de vectores de soporte son:

Efectivo en espacios de gran dimensión.

Sigue siendo eficaz en los casos en que el número de dimensiones es mayor que el número de muestras. (Bache, 2013)

Utiliza un subconjunto de puntos de entrenamiento en la función de decisión (llamados vectores de soporte), por lo que también es eficiente en la memoria.

Versátil: se pueden especificar diferentes funciones del núcleo para la función de decisión. Se proporcionan núcleos comunes, pero también es posible especificar núcleos personalizados.

Las desventajas de las máquinas de vectores de soporte incluyen:

Si el número de características es mucho mayor que el número de muestras, evite el ajuste excesivo al elegir las funciones del Kernel y el término de regularización es crucial.

Las SVM no proporcionan directamente estimaciones de probabilidad, estas se calculan mediante una costosa validación cruzada de cinco veces (consulte Puntuaciones y probabilidades, a continuación).

Las máquinas de vectores de soporte en scikit-learn admiten vectores de muestra densos (`numpy.ndarray` y convertibles a eso por `numpy.asarray`) y dispersos (cualquier `scipy.sparse`) como entrada. Sin embargo, para usar una SVM para hacer predicciones para datos escasos, debe haber encajado en dichos datos. Para un rendimiento óptimo, use `numpy.ndarray` (denso) ordenado en C o `scipy.sparse.csr_matrix` (disperso) con `dtype = float64`. (Akbani, 2004).

El objetivo del algoritmo SVM es encontrar un hiperplano que separe de la mejor forma posible dos clases diferentes de puntos de datos. “De la mejor forma posible” implica el hiperplano con el margen más amplio entre las dos clases, representado por los signos más y menos en la siguiente figura. El margen se define como la anchura máxima de la región paralela al hiperplano que no tiene puntos de datos interiores. El algoritmo solo puede encontrar este hiperplano en problemas que permiten separación lineal; en la mayoría de los problemas prácticos, el algoritmo maximiza el margen flexible permitiendo un pequeño número de clasificaciones erróneas.

Los vectores de soporte hacen referencia a un subconjunto de las observaciones de entrenamiento que identifican la ubicación del hiperplano de separación. El algoritmo SVM estándar está formulado para problemas de clasificación binaria; los problemas multiclase normalmente se reducen a una serie de problemas binarios.

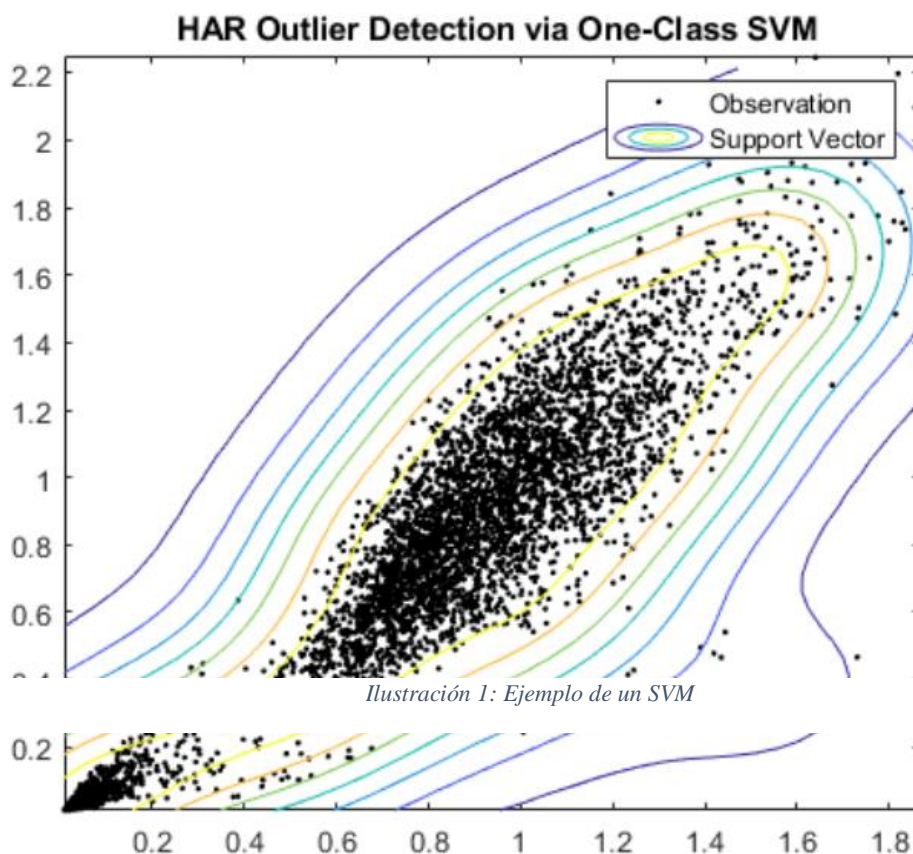


Ilustración 1: Ejemplo de un SVM

Codificación y ejecución del script

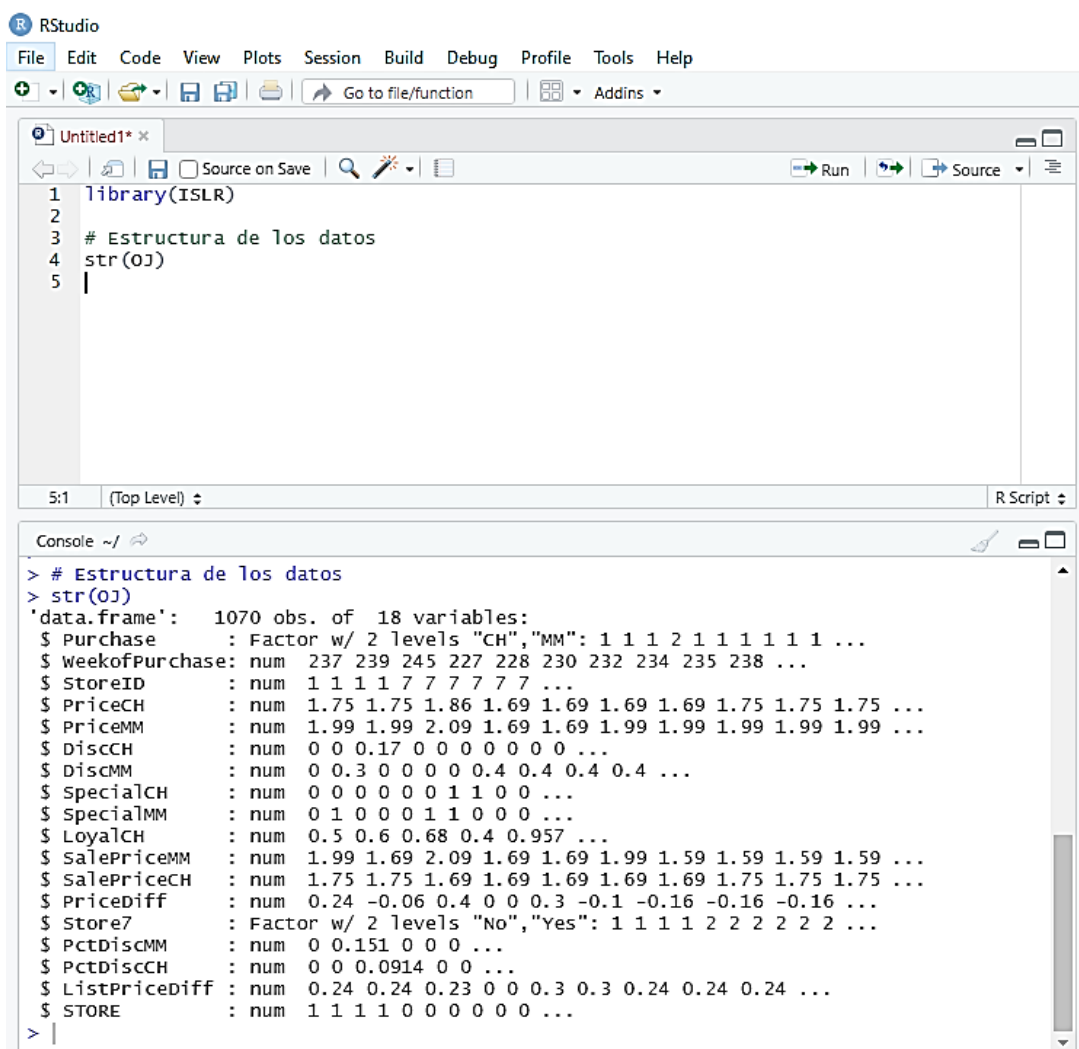
Para la ejecución de ejemplo de clasificación se utilizarán varios datos entre ellos *oj* pertenecientes al paquete *ISLR*.

Contiene información sobre compra de dos tipos de bebida (Citrus Hill y Minute Maid Orange Juice) por parte de 1070 clientes (las variables registran distintas características del cliente y el producto). Generaremos modelos basados en SVM con tres tipos de kernel: lineal, polinómico y radial, que predigan qué tipo de bebida (Purchase) compra el consumidor, en función del conjunto de predictores.

Proceso

Paso 1

Se realiza la instalación de la librería *ISLR*, se procede al llamado a los datos *OJ* con la sentencia *str*, en la cual aparecerán los datos de cada uno de los productos



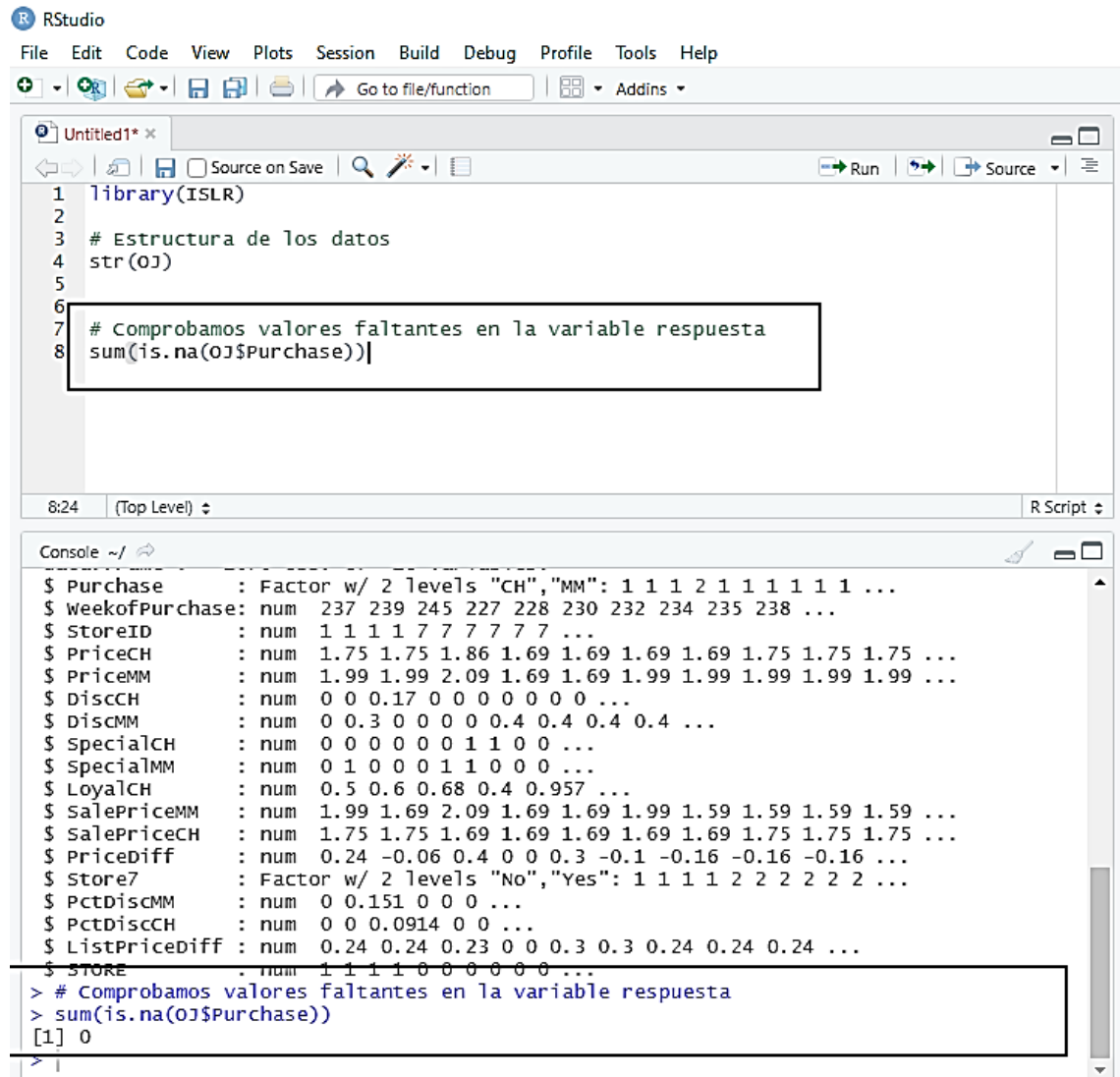
```
1 library(ISLR)
2
3 # Estructura de los datos
4 str(OJ)
5 |
```

```
> # Estructura de los datos
> str(OJ)
'data.frame': 1070 obs. of 18 variables:
 $ Purchase      : Factor w/ 2 levels "CH","MM": 1 1 1 2 1 1 1 1 1 1 ...
 $ WeekofPurchase: num  237 239 245 227 228 230 232 234 235 238 ...
 $ StoreID       : num  1 1 1 1 7 7 7 7 7 7 ...
 $ PriceCH       : num  1.75 1.75 1.86 1.69 1.69 1.69 1.69 1.75 1.75 1.75 ...
 $ PriceMM       : num  1.99 1.99 2.09 1.69 1.69 1.69 1.99 1.99 1.99 1.99 ...
 $ DiscCH        : num  0 0 0.17 0 0 0 0 0 0 0 ...
 $ DiscMM        : num  0 0.3 0 0 0 0 0.4 0.4 0.4 0.4 ...
 $ SpecialCH     : num  0 0 0 0 0 0 1 1 0 0 ...
 $ SpecialMM     : num  0 1 0 0 0 1 1 0 0 0 ...
 $ LoyalCH       : num  0.5 0.6 0.68 0.4 0.957 ...
 $ SalePriceMM   : num  1.99 1.69 2.09 1.69 1.69 1.99 1.59 1.59 1.59 1.59 ...
 $ SalePriceCH   : num  1.75 1.75 1.69 1.69 1.69 1.69 1.69 1.75 1.75 1.75 ...
 $ PriceDiff     : num  0.24 -0.06 0.4 0 0 0.3 -0.1 -0.16 -0.16 -0.16 ...
 $ Store7       : Factor w/ 2 levels "No","Yes": 1 1 1 1 2 2 2 2 2 2 ...
 $ PctDiscMM     : num  0 0.151 0 0 0 ...
 $ PctDiscCH     : num  0 0 0.0914 0 0 ...
 $ ListPriceDiff : num  0.24 0.24 0.23 0 0 0.3 0.3 0.24 0.24 0.24 ...
 $ STORE        : num  1 1 1 1 0 0 0 0 0 0 ...
```

Ilustración 2: Instalación de la librería *ISLR*

Paso 2

Comprobar los valores faltantes dentro del conjuntos de valores previamente mostrados para lo cual se procede a la función suma la cual contara los valores faltantes.



```

1 library(ISLR)
2
3 # Estructura de los datos
4 str(OJ)
5
6
7 # Comprobamos valores faltantes en la variable respuesta
8 sum(is.na(OJ$Purchase))

```

```

$ Purchase      : Factor w/ 2 levels "CH","MM": 1 1 1 2 1 1 1 1 1 1 ...
$ WeekofPurchase: num  237 239 245 227 228 230 232 234 235 238 ...
$ StoreID       : num  1 1 1 1 7 7 7 7 7 7 ...
$ PriceCH       : num  1.75 1.75 1.86 1.69 1.69 1.69 1.69 1.75 1.75 1.75 ...
$ PriceMM       : num  1.99 1.99 2.09 1.69 1.69 1.69 1.99 1.99 1.99 1.99 ...
$ DiscCH        : num  0 0 0.17 0 0 0 0 0 0 0 ...
$ DiscMM        : num  0 0.3 0 0 0 0 0.4 0.4 0.4 0.4 ...
$ SpecialCH     : num  0 0 0 0 0 0 1 1 0 0 ...
$ SpecialMM     : num  0 1 0 0 0 1 1 0 0 0 ...
$ LoyalCH       : num  0.5 0.6 0.68 0.4 0.957 ...
$ SalePriceMM   : num  1.99 1.69 2.09 1.69 1.69 1.99 1.59 1.59 1.59 1.59 ...
$ SalePriceCH   : num  1.75 1.75 1.69 1.69 1.69 1.69 1.69 1.75 1.75 1.75 ...
$ PriceDiff     : num  0.24 -0.06 0.4 0 0 0.3 -0.1 -0.16 -0.16 -0.16 ...
$ Store7        : Factor w/ 2 levels "No","Yes": 1 1 1 1 2 2 2 2 2 2 ...
$ PctDiscMM     : num  0 0.151 0 0 0 ...
$ PctDiscCH     : num  0 0 0.0914 0 0 ...
$ ListPriceDiff : num  0.24 0.24 0.23 0 0 0.3 0.3 0.24 0.24 0.24 ...
$ STORE        : num  1 1 1 1 0 0 0 0 0 0 ...
> # Comprobamos valores faltantes en la variable respuesta
> sum(is.na(OJ$Purchase))
[1] 0

```

Ilustración 3: Valores faltantes

Paso 3

Se necesita la instalación de la librería **ggplot2**, la cual cumplirá la función para declarar más de un valor, en dicha se guardarán los valores de los ejes x, y; cada uno representando a cada una de las bebidas, geom_bar se encargará de contabilizar cada uno de los valores.

Definen el título del gráfico, añadido de definirán los colores de cada una de las barras, guardados cada detalle de la gráfica, y procede a mostrar los resultados

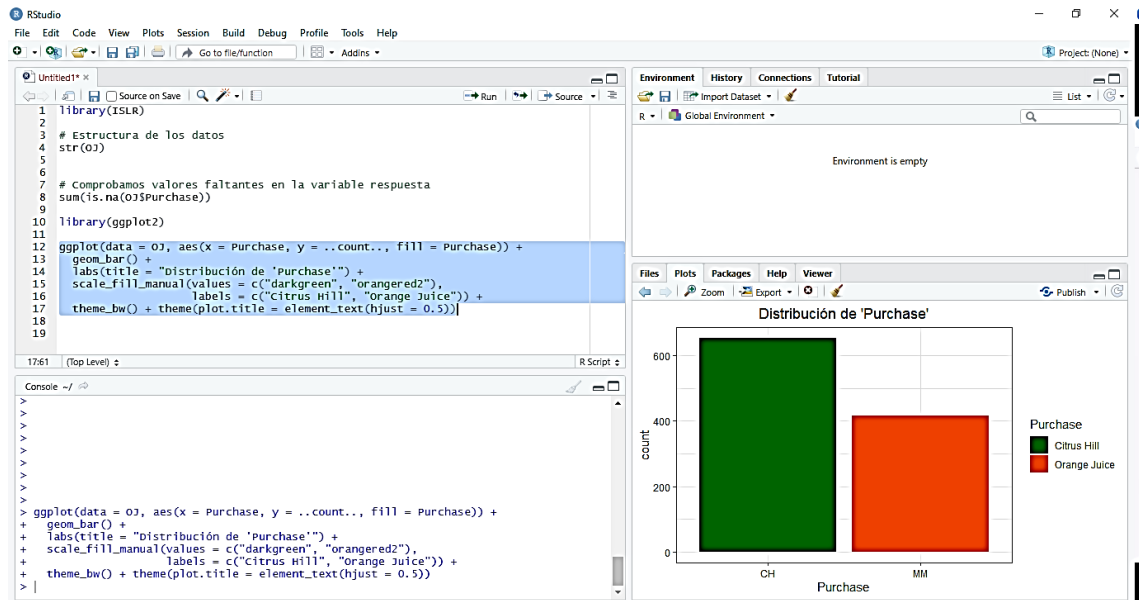


Ilustración 4: Librería ggplot2

Paso 4

Con la función **table** procedemos a mostrar el valor máximo alcanzado con cada uno de los valores dentro de la tabla purchase



Ilustración 5: Valor máximo

Paso 5

Se descarga la librería **dplyr**, la cual incluye un conjunto de comandos que coinciden con las acciones más comunes que se realizan sobre un conjunto de datos. Mostrando los porcentajes perdidos dentro de la tabla purchase tomando en cuenta dos dígitos de redondeo


```
19 table(OJ$Purchase)
20
21
22 library(dplyr)
23 prop.table(table(OJ$Purchase)) %>% round(digits = 2)
24
25
```

22:1 (Top Level) R Script

Console ~/

```
>
>
>
>
>
>
>
>
>
>
>
> library(dplyr)
> prop.table(table(OJ$Purchase)) %>% round(digits = 2)
```

CH	MM
0.61	0.39

```
>
```

Ilustración 6: Librería dplyr

Para que los modelos generados sean útiles, el porcentaje de aciertos en cuanto a la clasificación de las observaciones ha de superar un nivel mínimo, en este caso, el que se obtendría si la predicción de todas las observaciones se correspondiera con la clase mayoritaria. La clase mayoritaria (moda) en este caso es la bebida CH con el 61% de las compras. Este será el nivel basal a superar por el modelo (este es el porcentaje mínimo de aciertos si siempre se predijera CH). (Recalcular este valor con los datos de entrenamiento).

Antes de proceder a generar los modelos, dividimos el set de datos en un grupo de entrenamiento (para el ajuste de los modelos) y otro de test (para la evaluación de los mismos). Esta división dependerá de la cantidad de observaciones con las que contemos y la seguridad con la que queramos obtener la estimación del test error. En este ejemplo se opta por una división 80%-20%.

Paso 6

Se instala la librería **caret** la cual es un conjunto de funciones que intentan agilizar el proceso de creación de modelos predictivos se utiliza `datosOJ_train <- OJ[train,]` y luego el **dim** para la dimensión de los datos

```
24  
25  
26 library(caret)  
27  
28 # Índices observaciones de entrenamiento  
29 set.seed(123)  
30 train <- createDataPartition(y = OJ$Purchase, p = 0.8, list = FALSE, times = 1)  
31  
32 # Datos entrenamiento  
33 datosOJ_train <- OJ[train, ]  
34 dim(datosOJ_train)  
35
```

35:1 (Top Level) R Script

Console ~/
>
>
>
>
>
> library(caret)
>
> # Índices observaciones de entrenamiento
> set.seed(123)
> train <- createDataPartition(y = OJ\$Purchase, p = 0.8, list = FALSE, times = 1)
>
> # Datos entrenamiento
> datosOJ_train <- OJ[train,]
> dim(datosOJ_train)
[1] 857 18

Ilustración 7: Librería caret

Paso 7

Se utiliza `datosOJ_test <- OJ[-train,]` y luego el **dim** para la dimensión de los datos de test

```
36  
37  
38  
39  
40 datosOJ_test <- OJ[-train, ]  
41 dim(datosOJ_test)  
42  
43
```

39:1 | (Top Level) ↕ R Script ↕

Console ~/ ➡

```
>  
>  
>  
>  
>  
>  
>  
>  
>  
>  
>  
> datosOJ_test <- OJ[-train, ]  
> dim(datosOJ_test)  
[1] 213 18  
>
```

Ilustración 8: Dimensión de prueba

Conclusiones

Se estudió el método de máquinas de vectores de soporte como alternativa a los modelos de regresión logística conservadores y se comparó su desempeño con los conjuntos de datos de crédito real. Especialmente en combinación con el kernel no lineal, SVM demostró ser un enfoque competitivo y proporcionó una ligera ventaja sobre el modelo de regresión logística.

Otro argumento más serio contra SVM surge del hecho de que es muy difícil usarlo como método independiente. A pesar de un gran esfuerzo, no pude implementar un proceso de selección de características confiable de acuerdo con las técnicas recomendadas en la literatura (que se resumieron en la sección Selección de características. Finalmente, tuve que desviarme al enfoque alternativo, utilizando pruebas estadísticas rigurosas en relación con la regresión logística, para seleccionar las variables explicativas más significativas estadísticamente para construir el modelo. Más importante aún, este es el único método mencionado explícitamente en la literatura disponible para mí, para cuantificarle efecto de cada variable en el modelo de máquinas de vectores de soporte. Esta desventaja está ligeramente equilibrada por el hecho de que SVM tiende a funcionar mejor en la forma del modelo sin restricciones.

Referencias

- Akbani, R. K. (2004). Aplicación de máquinas de vectores de soporte a conjuntos de datos desequilibrados. En N. Japkowicz. Springer: En Machine Learning.
- Bache, K. y. (2013). *M. Iris data set*. Obtenido de <http://archive.ics.uci.edu/ml/datasets/Iris>
- Baesens, B. V. (2003). Análisis comparativo de los algoritmos de clasificación de vanguardia para la calificación crediticia. *Revista de la Sociedad de Investigación Operativa* 54, 627-635.