SVM

María Bustamante, Nicolás Jadán

SVM

1. Recopilar datos y transformar

```
libraries <- c("reshape2", "ggplot2", "kernlab" ,"caret")
check.libraries <- is.element(libraries, installed.packages()[, 1])==FALSE
libraries.to.install <- libraries[check.libraries]
if (length(libraries.to.install!=0)) {
  install.packages(libraries.to.install)
}

success <- sapply(libraries,require, quietly = FALSE,  character.only = TRUE)
```

```
Loading required package: reshape2

Loading required package: ggplot2

Loading required package: kernlab


Attaching package: 'kernlab'

The following object is masked from 'package:ggplot2':

    alpha

Loading required package: caret

Loading required package: lattice
```

```
if(length(success) != length(libraries)) {stop("A package failed to return a success in re
```

Explicación:

En esta sección se comprueba si las bibliotecas que se necesitan para esta tarea están instaladas y las instala automáticamente si no lo están. Después, comprueba si todas las bibliotecas se cargaron corectamente y muestra un mensaje de errores si alguna no se instaló bien.

```
datos<-read.csv("./cancer.csv")
```

Explicación:

Se carga el conjunto de datos con el que se va a trabajar. Dicho conjunto de datos, consta de 31 variables, la primera es la clasificación del tumor (benigno o maligno), mientras que las variables restantes, son características de las células, producto de un procesado de imagen.

```
datos$diagnostico <- as.factor(datos$diagnostico)
clase <- datos$diagnostico
```

Explicación:

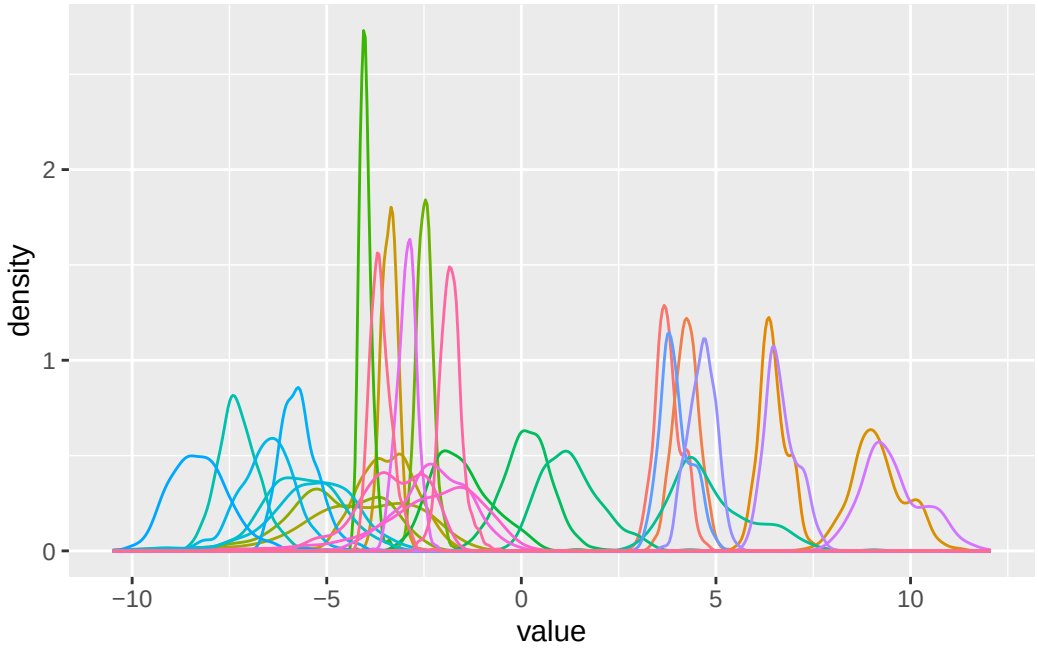
En esta parte se convierte la variable categórica a factoe y se guarda en la misma variable para poder utilizarla en los siguientes pasos. En la segunda línea se extrae la columna de diagnóstico y se un objeto llamado clase.

```
Col<-colnames(datos)[2:31]
X<-datos[,Col]
X.melt<-melt((log2(X)))
```

```
No id variables; using all as measure variables
```

```
p <- ggplot(aes(x=value,colour=variable), data=X.melt)
p + geom_density(show.legend = F)
```

```
Warning: Removed 78 rows containing non-finite values (`stat_density()`).
```



Explicación:

- `Col<-colnames(datos)[2:31]`: En esta línea, se crea un nuevo vector (col) que abarca las columnas desde la posición 2 hasta la 31.
- `X<-datos[,Col]`: Se crea un nuevo objetoo que contiene todas las filas y solo las columnas especificadas en el vector **`Col`** del objeto de datos.
- `X.melt<-melt((log2(X)))`: Melt se usa para transformar el objeto X en un formato largo o longitudinal. También se aplica la función **`log2()`** a los valores de **`X`** antes de la transformación. Con la función **`melt()`** se facilita el uso de los datos en visualizaciones posteriores.
- `p <- ggplot(aes(x=value,colour=variable), data=X.melt)`: En esta línea, se crea un objeto ggplot llamado p. Se especifica que el eje x del gráfico estará representado por la columna value en el objeto X.melt, y el color de las líneas de densidad se basará en la columna variable en X.melt.
- `p + geom_density(show.legend = F)`: Finalmente, se agrega una capa de densidad al gráfico p utilizando geom_density(). Esto generará las líneas de densidad en el gráfico. El argumento show.legend = F se utiliza para evitar que se muestre la leyenda en el gráfico.
- Gráfica: Las curvas de densidad representan la distribución de valores para una variable en específico. Las variables de cada curva se identifican por el color de la curva. Al trazar

las curvas de densidad, se puede visualizar la distribución y la forma de las variables seleccionadas. Las áreas más altas y anchas indican una mayor concentración de valores en esas áreas. Por el contrario, los rangos más bajos y más estrechos indican una menor concentración de valores.

```
X.log<-log2(X)
datos.log<-cbind(X,clase)
class(datos.log)
```

```
[1] "data.frame"
```

Explicación:

- `X.log<-log2(X)`: Se aplica la función log2() a la matriz de datos X para calcular el logaritmo base 2 de cada valor en X. Esto se hace para transformar los valores de X utilizando el logaritmo.
- `datos.log <- cbind(X, clase)`:cbind se usa para combinar las columnas de X y clase en un solo objeto de datos, en donde X tiene las variables de entrada mientras que clase contiene las de salida corrspondiente a cada observación. En resumen en "datos.log", "X.log" se ubicará en las primeras columnas, seguido por la columna "clase".
- `class(datos.log)`: se devuelve la clse de 'datos.log'.

2. División de datos para entrenamiento y prueba

```
datos2<-colnames(datos[1:31])
datos2<-datos[,datos2]
datos$diagnostico<-as.factor(datos2$diagnostico)
```

Explicación:

En este código se crea un vector de datos que contiene las columnas desde la 1 a la 31, haciendo uso de la función colnames. Luego, se convierte la columna "diagnostico" del objetodatos en un factor utilizando la columna correspondiente en datos2.

```
set.seed(123456)

datos2[,2:31]<-as.data.frame(scale(datos2[,2:31]))

levels(datos2)<-c("B","M")
```

```r
entrenamiento<-sample(nrow(datos2),size=nrow(datos2)*0.7)

entrenamiento.data<-datos2[entrenamiento,]
prueba.data<-datos2[-entrenamiento,]
```

Explicación:

- `set.seed(123456)`: Se fija la semilla para asegurar que los resultados sean los mismos cada vez que se ejecute el código.
- `datos2[,2:31]`: La función scale() realiza una estandarización, lo que significa que centra los valores alrededor de su media y los escala por su desviación estándar. La función as.data.frame() se utiliza para convertir los resultados de la estandarización nuevamente en un objeto de datos.
- `levels(datos2)<-c("B","M"):` Se establecen los niveles de la variable de interés siendo Benigno (B) y Maligno (M).
- Finalmente se dividen los datos en 70% para entrenamiento y 30% para prueba.

3. Entrenamiento del modelo

```r
clasifier.lineal <- ksvm(diagnostico ~ ., data = entrenamiento.data, kernel = "vanilladot"
```

```
Setting default kernel parameters
```

```r
clasifier.gauss <- ksvm(diagnostico ~ ., data = entrenamiento.data, kernel = "rbfdot")
```

Explicación:

Este código realiza dos clasificadores tilizando el algoritmo de SVM con diferentes kernels.

- `clasifier.lineal`: En esta línea, se crea un clasificador SVM lineal utilizando la función ksvm() . El argumento diagnostico ~ . especifica que se utiliza la columna diagnostico como variable objetivo y todas las demás columnas del objeto de datos entrenamiento.data como variables predictoras. El kernel utilizado es "vanilladot", que es un kernel lineal.
- `clasifier.gauss`: En esta línea, se crea otro clasificador SVM utilizando la misma estructura que en el paso anterior, pero con un kernel diferente. En este caso, se utiliza el kernel "rbfdot", que es un kernel gaussiano

```
clasifier.lineal
```

```
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 1

Linear (vanilla) kernel function.

Number of Support Vectors : 33

Objective Function Value : -21.2999
Training error : 0.012563
```

Explicación:

- C-svc significa que se está utilizando un SVM de clasificación con función de costo. El parámetro C=1 es un factor de regularización que controla la penalización por clasificar incorrectamente puntos de datos en el algoritmo. En este caso indica un factor de regularización moderado.
- linear (vanilla): El kernel lineal es una función que permite realizar la clasificación en un espacio de características lineal.
- Número de Vectores de Soporte (33): Indica el número de vectores de soporte encontrados durante el entrenamiento del modelo. Los vectores de soporte son los puntos de datos que influyen en la definición del hiperplano de separación entre las clases.
- Valor de la Función Objetivo (-21.2999): Este valor representa la función objetivo del modelo SVM. En general, se busca minimizar esta función durante el entrenamiento del modelo.
- Error de entrenamiento (0.012563): Este valor indica el error de entrenamiento del modelo SVM, es decir, el porcentaje de puntos de datos de entrenamiento que se clasificaron incorrectamente. En este caso, el error de entrenamiento es del 1.26%.

En conclusión el valor de la función objetivo del modelo es de -21.2999 y el error de entrenamiento obtenido es de 0.012563, lo que muestra que el modelo ha logrado una buena capacidad de clasificación en los datos de entrenamiento.

```
clasifier.gauss
```

```
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 1

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.0405393061667693

Number of Support Vectors : 107

Objective Function Value : -47.3373
Training error : 0.015075
```

Explicación:

- El kernel RBF es una función que permite realizar clasificación en espacios de características no lineales.
- Hiperparámetro sigma: En este caso, se utilizó un valor de sigma igual a 0.0405, que implica que los puntos de datos cercanos tienen una influencia más fuerte en la clasificación.
- Número de vectores de soporte: En este caso, se encontraron 107 vectores de soporte durante el entrenamiento del modelo. Estos vectores de soporte son los puntos clave para la clasificación y se utilizan para definir el hiperplano de separación en la máquina de vectores.
- Valor de la función objetivo: El valor de la función fue de -47,33, lo que indica que se obtuvo un buen ajuste del modelo a los datos de entrenamiento.
- Error de entrenamiento: El error de entrenamiento fue bajo, es decir que el 1.50075% de puntos de datos de entrenamiento fueron clasificados incorrectamente por el modelo.

4. Evaluación del rendimiento del modelo

```r
prediction.linear <- predict(clasifier.lineal, newdata = prueba.data)
res.linear <- table(prediction.linear, prueba.data$diagnostico)

prediction.gauss <- predict(clasifier.gauss, newdata = prueba.data)
res.gauss <- table(prediction.gauss, prueba.data$diagnostico)
```

Explicación:

El código realiza la predicción utilizando los clasificadores SVM lineal y SVM gaussiano en el conjunto de datos de prueba. Luego, se crea una tabla de contingencia para evaluar la precisión de las predicciones realizadas por cada clasificador. Las tablas de contingencia (**res.linear** y **res.gauss**) permiten comparar las predicciones con las etiquetas de clase reales y analizar la precisión de los clasificadores en la clasificación de los datos de prueba.

```
cmatrix1 <- confusionMatrix(res.linear, positive = levels(prueba.data$diagnostico))
cmatrix1
```

```
Confusion Matrix and Statistics


prediction.linear   B   M
                B 107   3
                M   2  59

Warning in cbind(format(overallNames, justify = "right"), overallText): number
of rows of result is not a multiple of vector length (arg 1)


               Accuracy : 0.9708
                 95% CI : (0.9331, 0.9904)
    No Information Rate : 0.6374
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9365

 Mcnemar's Test P-Value : 1

            Sensitivity :  1
            Specificity : NA
         Pos Pred Value : NA
         Neg Pred Value : NA
             Prevalence :  1
         Detection Rate :  1
   Detection Prevalence :  1
      Balanced Accuracy : NA

       'Positive' Class : B
              Accuracy : M
```

```r
cmatrix2 <- confusionMatrix(res.gauss, positive = levels(prueba.data$diagnostico))
cmatrix2
```

```
Confusion Matrix and Statistics


prediction.gauss   B   M
               B 107   3
               M   2  59

Warning in cbind(format(overallNames, justify = "right"), overallText): number
of rows of result is not a multiple of vector length (arg 1)


               Accuracy : 0.9708
                 95% CI : (0.9331, 0.9904)
    No Information Rate : 0.6374
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9365

 Mcnemar's Test P-Value : 1

            Sensitivity :  1
            Specificity : NA
         Pos Pred Value : NA
         Neg Pred Value : NA
             Prevalence :  1
         Detection Rate :  1
   Detection Prevalence :  1
      Balanced Accuracy : NA

       'Positive' Class : B
               Accuracy : M
```

Explicación:

En esta parte, se utiliza la función **`confusionMatrix`** del paquete "caret" para calcular las matrices de confusión para cada clasificador. En este caso, se utiliza **`levels(prueba.data$diagnostico)`**

para obtener los niveles únicos de la variable objetivo en los datos de prueba y asegurarse de que el valor positivo esté correctamente especificado.

Finalmente, se almacenan las matrices de confusión en las variables **cmatrix1** y **cmatrix2**, respectivamente, y se imprimen en la consola para visualizar los resultados.

Opcional: Validación cruzada

```
model.5v.linear <- train(diagnostico ~ ., entrenamiento.data, method='svmLinear',
                trControl= trainControl(method='cv', number=5),
                tuneGrid= NULL, tuneLength=10 ,trace = FALSE)

# plot(model.5v, alpha=0.6)
summary(model.5v.linear)
```

```
Length  Class   Mode
     1   ksvm     S4
```

```
prediction <- predict(model.5v.linear, prueba.data)                              # predict
res.linear.2<-table(prediction, prueba.data$diagnostico)

# predict can also return the probability for each class:
cmatrix <- confusionMatrix(res.linear.2, positive = levels(prueba.data$diagnostico))
cmatrix
```

```
Confusion Matrix and Statistics


prediction   B   M
         B 107   3
         M   2  59

Warning in cbind(format(overallNames, justify = "right"), overallText): number
of rows of result is not a multiple of vector length (arg 1)


               Accuracy : 0.9708
                 95% CI : (0.9331, 0.9904)
    No Information Rate : 0.6374
```

```
   P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9365

 Mcnemar's Test P-Value : 1

            Sensitivity :  1
            Specificity : NA
         Pos Pred Value : NA
         Neg Pred Value : NA
             Prevalence :  1
         Detection Rate :  1
   Detection Prevalence :  1
      Balanced Accuracy : NA

       'Positive' Class : B
              Accuracy : M
```

Explicación:

Se entrena un modelo de Máquina de Vectores de Soporte (SVM) lineal utilizando la función **`train`** del paquete "caret".

Se utiliza la validación cruzada de 5 pliegues para evaluar el rendimiento del modelo.

Obtenemos un resumen del modelo entrenado utilizando la función **`summary`**.

Se realizan predicciones en los datos de prueba utilizando el modelo entrenado con la función **`predict`**.

Crea una tabla de contingencia para comparar las predicciones con los valores reales.

Utiliza la función **`confusionMatrix`** para calcular la matriz de confusión y evaluar el rendimiento del modelo.

La matriz de confusión se almacena en la variable **`cmatrix`**.

En resumen, el código entrena un modelo SVM lineal, realiza predicciones en los datos de prueba y calcula la matriz de confusión para evaluar el rendimiento del modelo

Resultados:

En este caso, la precisión es 0.9708, lo que indica que el modelo clasificó correctamente el 97.08% de las instancias.

La sensibilidad tiene un valor de 1, lo que significa que todas las instancias de la clase "B" fueron clasificadas correctamente.