Code

Clasificación usando Vecino más Cercano (kNN)

1. Introducción

- Vimos que el algoritmo kNN sirve para hacer clasificación.
- Partimos de un outcome de interés.
- Que corresponde a una variable categórica: por ejemplo, si un correo es spam o no.
- Clasificamos cada caso de acuerdo a otras características.
- Buscando los k vecinos más cercanos.
- Cada caso es asignado a la categoría a la que pertenecen la mayoría de vecinos.
- Veremos a continuación cómo funciona esto en la práctica.
- Usaremos el algoritmo kNN para el diagnístico de cáncer de seno

2. Diagnóstico de cáncer de seno con el algoritmo kNN

- La detección temprana de cáncer de seno se hace examinando el tejido del seno, para buscar bultos o masas anormales.
- Si se encuentra un bulto, se realiza una bipsia en la que se extrae una muestra de tejido por medio de una jeringa.
- La muestra de células es analizada con microspcopio para determinar si la masa probablemente es benigna o maligna.
- Con técnicas de machine learning se podría automatizar el proceso de detección de células cancerosas, lo cual sería beneficioso para el sistema de salud.
- Los médicos podrían pasar más tiempo tratando la enfermedad si el proceso de detección se automatiza.
- Además el proceso automatizado es más eficiente si es capaz de eliminar el componente subjetivo de los humanos.

3. Datos

- Para ilustrar el procedimiento usaremos los datos Wisconsin Breast Cancer Diagnostic, de la Universidad de Wisconsin.
- Se incluyen medidas obtenidas de imágenes digitalizadas de masas de senos extraídas con agujas.
- Los valores representan características del núcleo celular presente en la imágenes.
- Los datos incluyen 569 ejemplos de biopsias cancerosas, cada una con 32 características.
- Se incluye un número de identificación.
- El diagnóstico se codifica como benigno “B” o maligno “M”.
- Las restantes 30 características incluyen la media, error estándar y valor máximo de 10 características del núcleo celular, como son:
    - Radio
    - Textura
    - Perímetro
    - Área
    - Suavidad
    - Compacidad
    - Concavidad
    - Puntos cóncavos
    - Simetría
    - Dimensión fractal

- Para clasificar una muestra como benigna o maligna de acuerdo con sus características, trabajaremos con la base `wisc_bc_data.csv` .

- Importemos la base:

Hide

Hide

```
wbcd <- read.csv("wisc_bc_data.csv", stringsAsFactors=FALSE)
```

- Podemos darle un primer vistazo con la función `str()` :

Hide

Hide

```
str(wbcd)
```

```
'data.frame':   569 obs. of  32 variables:
 $ id                      : int  842302 842517 84300903 84348301 84358402 843786 8443
59 84458202 844981 84501001 ...
 $ diagnosis               : chr  "M" "M" "M" "M" ...
 $ radius_mean             : num  18 20.6 19.7 11.4 20.3 ...
 $ texture_mean            : num  10.4 17.8 21.2 20.4 14.3 ...
 $ perimeter_mean          : num  122.8 132.9 130 77.6 135.1 ...
 $ area_mean               : num  1001 1326 1203 386 1297 ...
 $ smoothness_mean         : num  0.1184 0.0847 0.1096 0.1425 0.1003 ...
 $ compactness_mean        : num  0.2776 0.0786 0.1599 0.2839 0.1328 ...
 $ concavity_mean          : num  0.3001 0.0869 0.1974 0.2414 0.198 ...
 $ concave.points_mean     : num  0.1471 0.0702 0.1279 0.1052 0.1043 ...
 $ symmetry_mean           : num  0.242 0.181 0.207 0.26 0.181 ...
 $ fractal_dimension_mean : num  0.0787 0.0567 0.06 0.0974 0.0588 ...
 $ radius_se               : num  1.095 0.543 0.746 0.496 0.757 ...
 $ texture_se              : num  0.905 0.734 0.787 1.156 0.781 ...
 $ perimeter_se            : num  8.59 3.4 4.58 3.44 5.44 ...
 $ area_se                 : num  153.4 74.1 94 27.2 94.4 ...
 $ smoothness_se           : num  0.0064 0.00522 0.00615 0.00911 0.01149 ...
 $ compactness_se          : num  0.049 0.0131 0.0401 0.0746 0.0246 ...
 $ concavity_se            : num  0.0537 0.0186 0.0383 0.0566 0.0569 ...
 $ concave.points_se       : num  0.0159 0.0134 0.0206 0.0187 0.0188 ...
 $ symmetry_se             : num  0.03 0.0139 0.0225 0.0596 0.0176 ...
 $ fractal_dimension_se    : num  0.00619 0.00353 0.00457 0.00921 0.00511 ...
 $ radius_worst            : num  25.4 25 23.6 14.9 22.5 ...
 $ texture_worst           : num  17.3 23.4 25.5 26.5 16.7 ...
 $ perimeter_worst         : num  184.6 158.8 152.5 98.9 152.2 ...
 $ area_worst              : num  2019 1956 1709 568 1575 ...
 $ smoothness_worst        : num  0.162 0.124 0.144 0.21 0.137 ...
 $ compactness_worst       : num  0.666 0.187 0.424 0.866 0.205 ...
 $ concavity_worst         : num  0.712 0.242 0.45 0.687 0.4 ...
 $ concave.points_worst    : num  0.265 0.186 0.243 0.258 0.163 ...
 $ symmetry_worst          : num  0.46 0.275 0.361 0.664 0.236 ...
 $ fractal_dimension_worst: num  0.1189 0.089 0.0876 0.173 0.0768 ...
```

- También podemos usar la función `head()`:

Hide

Hide

```
head(wbcd)
```

	id	diagnosis	radius_mean	texture_mean

	<int>	<chr>	<dbl>	<dbl>
1	842302	M	17.99	10.38
2	842517	M	20.57	17.77
3	84300903	M	19.69	21.25
4	84348301	M	11.42	20.38
5	84358402	M	20.29	14.34
6	843786	M	12.45	15.70

6 rows | 1-5 of 32 columns

- Naturalmente, `id` es un identificador que nos dice poco sobre el estátus de la muestra.
- Debemos excluir dicha variable del modelo. Redefinimos `wbcd` excluyendo su primera columna

Hide

Hide

```
wbcd <- wbcd[-1]
```

- El outcome de interés es `diagnosis` . Es el que queremos predecir. Exploremos esta variables con la función `table()` :

Hide

Hide

```
table(wbcd$diagnosis)
```

```
  B   M 
357 212 
```

- De los 569 casos, 357 son benignos frente a 212 malignos.
- Muchos algoritmos requieren que la característica de interés sea un factor. Codifiquemos `diagnosis` con tal propósito:

Hide

Hide

```
wbcd$diagnosis <- factor(wbcd$diagnosis, levels = c("B", "M"), labels = c("Benign", "
Malign"))
```

- Con la función `prop.table` obtenemos la proporción para cada caso:

Hide

Hide

```
round(prop.table(table(wbcd$diagnosis))*100, digits=1)
```

```
Benign Malign
  62.7   37.3
```

- Las demás variables son numéricas. Las podemos explorar con `summary()`:

Hide

Hide

```
summary(wbcd[2:31])
```

```
 radius_mean      texture_mean   perimeter_mean     area_mean
Min.   : 6.981   Min.   : 9.71   Min.   : 43.79   Min.   : 143.5
1st Qu.:11.700   1st Qu.:16.17   1st Qu.: 75.17   1st Qu.: 420.3
Median :13.370   Median :18.84   Median : 86.24   Median : 551.1
Mean   :14.127   Mean   :19.29   Mean   : 91.97   Mean   : 654.9
3rd Qu.:15.780   3rd Qu.:21.80   3rd Qu.:104.10   3rd Qu.: 782.7
Max.   :28.110   Max.   :39.28   Max.   :188.50   Max.   :2501.0
smoothness_mean   compactness_mean  concavity_mean
Min.   :0.05263   Min.   :0.01938   Min.   :0.00000
1st Qu.:0.08637   1st Qu.:0.06492   1st Qu.:0.02956
Median :0.09587   Median :0.09263   Median :0.06154
Mean   :0.09636   Mean   :0.10434   Mean   :0.08880
3rd Qu.:0.10530   3rd Qu.:0.13040   3rd Qu.:0.13070
Max.   :0.16340   Max.   :0.34540   Max.   :0.42680
concave.points_mean symmetry_mean    fractal_dimension_mean
Min.   :0.00000     Min.   :0.1060   Min.   :0.04996
1st Qu.:0.02031     1st Qu.:0.1619   1st Qu.:0.05770
Median :0.03350     Median :0.1792   Median :0.06154
```

```
Mean   :0.04892     Mean   :0.1812   Mean   :0.06280
3rd Qu.:0.07400     3rd Qu.:0.1957   3rd Qu.:0.06612
Max.   :0.20120     Max.   :0.3040   Max.   :0.09744
  radius_se        texture_se      perimeter_se
Min.   :0.1115   Min.   :0.3602   Min.   : 0.757
1st Qu.:0.2324   1st Qu.:0.8339   1st Qu.: 1.606
Median :0.3242   Median :1.1080   Median : 2.287
Mean   :0.4052   Mean   :1.2169   Mean   : 2.866
3rd Qu.:0.4789   3rd Qu.:1.4740   3rd Qu.: 3.357
Max.   :2.8730   Max.   :4.8850   Max.   :21.980
   area_se         smoothness_se      compactness_se
Min.   :  6.802   Min.   :0.001713   Min.   :0.002252
1st Qu.: 17.850   1st Qu.:0.005169   1st Qu.:0.013080
Median : 24.530   Median :0.006380   Median :0.020450
Mean   : 40.337   Mean   :0.007041   Mean   :0.025478
3rd Qu.: 45.190   3rd Qu.:0.008146   3rd Qu.:0.032450
Max.   :542.200   Max.   :0.031130   Max.   :0.135400
 concavity_se     concave.points_se   symmetry_se
Min.   :0.00000   Min.   :0.000000   Min.   :0.007882
1st Qu.:0.01509   1st Qu.:0.007638   1st Qu.:0.015160
Median :0.02589   Median :0.010930   Median :0.018730
Mean   :0.03189   Mean   :0.011796   Mean   :0.020542
3rd Qu.:0.04205   3rd Qu.:0.014710   3rd Qu.:0.023480
Max.   :0.39600   Max.   :0.052790   Max.   :0.078950
fractal_dimension_se  radius_worst   texture_worst
Min.   :0.0008948    Min.   : 7.93   Min.   :12.02
1st Qu.:0.0022480    1st Qu.:13.01   1st Qu.:21.08
Median :0.0031870    Median :14.97   Median :25.41
Mean   :0.0037949    Mean   :16.27   Mean   :25.68
3rd Qu.:0.0045580    3rd Qu.:18.79   3rd Qu.:29.72
Max.   :0.0298400    Max.   :36.04   Max.   :49.54
perimeter_worst    area_worst     smoothness_worst
Min.   : 50.41   Min.   : 185.2   Min.   :0.07117
1st Qu.: 84.11   1st Qu.: 515.3   1st Qu.:0.11660
Median : 97.66   Median : 686.5   Median :0.13130
Mean   :107.26   Mean   : 880.6   Mean   :0.13237
3rd Qu.:125.40   3rd Qu.:1084.0   3rd Qu.:0.14600
Max.   :251.20   Max.   :4254.0   Max.   :0.22260
compactness_worst concavity_worst  concave.points_worst
Min.   :0.02729   Min.   :0.0000   Min.   :0.00000
1st Qu.:0.14720   1st Qu.:0.1145   1st Qu.:0.06493
Median :0.21190   Median :0.2267   Median :0.09993
Mean   :0.25427   Mean   :0.2722   Mean   :0.11461
3rd Qu.:0.33910   3rd Qu.:0.3829   3rd Qu.:0.16140
Max.   :1.05800   Max.   :1.2520   Max.   :0.29100
symmetry_worst   fractal_dimension_worst
Min.   :0.1565   Min.   :0.05504
1st Qu.:0.2504   1st Qu.:0.07146
Median :0.2822   Median :0.08004
Mean   :0.2901   Mean   :0.08395
```

```
3rd Qu.:0.3179   3rd Qu.:0.09208
Max.   :0.6638   Max.   :0.20750
```

- Claramente, las variables están en escalas diferentes. Deben rescalarse para hacerlas comparables.

4. Normalización de los Datos

- Recordemos que hay dos métodos para rescalar los datos:
    - Normalización min-max
    - Estandarización

- Veamos el caso de la normalización min-max.
- Creemos una función que nos permita normalizar los datos:

Hide

Hide

```
normalize <- function(x)
{
  return((x-min(x))/ (max(x)-min(x)))
}
```

- Probemos la función con el vector `c(1, 2, 3, 4, 5)` :

Hide

Hide

```
normalize(c(1,2,3,4,5))
```

```
[1] 0.00 0.25 0.50 0.75 1.00
```

- Funciona. Ahora apliquemos este procedimiento a las variables numéricas de `wbcd` .

- No tenemos que hacerlo una por una. Podemos usar la función `lapply()`, la cual aplica una función a cada elemento de una lista.
- Adicionalmente, le pedimos a R que el resultado de `lapply()` sea convertido en un data frame por medio de la función `as.data.frame()`:

Hide

Hide

```
wbcd_n <- as.data.frame(lapply(wbcd[2:31], normalize))
```

- Así, sobre cada una de las columnas 2-31 se ha aplicado la función `normalize`, el resultado se convierte en un data frame y guardamos ese objeto bajo el nombre `wbcd_n`.
- Podemos verificar si efectivamente se llevó a cabo la normalización:

Hide

Hide

```
summary(wbcd_n)
```

```
 radius_mean      texture_mean    perimeter_mean     area_mean
Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
1st Qu.:0.2233   1st Qu.:0.2185   1st Qu.:0.2168   1st Qu.:0.1174
Median :0.3024   Median :0.3088   Median :0.2933   Median :0.1729
Mean   :0.3382   Mean   :0.3240   Mean   :0.3329   Mean   :0.2169
3rd Qu.:0.4164   3rd Qu.:0.4089   3rd Qu.:0.4168   3rd Qu.:0.2711
Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
smoothness_mean  compactness_mean concavity_mean
Min.   :0.0000   Min.   :0.0000   Min.   :0.00000
1st Qu.:0.3046   1st Qu.:0.1397   1st Qu.:0.06926
Median :0.3904   Median :0.2247   Median :0.14419
Mean   :0.3948   Mean   :0.2606   Mean   :0.20806
3rd Qu.:0.4755   3rd Qu.:0.3405   3rd Qu.:0.30623
Max.   :1.0000   Max.   :1.0000   Max.   :1.00000
concave.points_mean symmetry_mean    fractal_dimension_mean
Min.   :0.0000      Min.   :0.0000   Min.   :0.0000
1st Qu.:0.1009      1st Qu.:0.2823   1st Qu.:0.1630
Median :0.1665      Median :0.3697   Median :0.2439
Mean   :0.2431      Mean   :0.3796   Mean   :0.2704
3rd Qu.:0.3678      3rd Qu.:0.4530   3rd Qu.:0.3404
Max.   :1.0000      Max.   :1.0000   Max.   :1.0000
  radius_se         texture_se      perimeter_se
Min.   :0.00000   Min.   :0.0000   Min.   :0.00000
1st Qu.:0.04378   1st Qu.:0.1047   1st Qu.:0.04000
```

```
Median :0.07702   Median :0.1653   Median :0.07209
Mean   :0.10635   Mean   :0.1893   Mean   :0.09938
3rd Qu.:0.13304   3rd Qu.:0.2462   3rd Qu.:0.12251
Max.   :1.00000   Max.   :1.0000   Max.   :1.00000
   area_se        smoothness_se    compactness_se
Min.   :0.00000   Min.   :0.0000   Min.   :0.00000
1st Qu.:0.02064   1st Qu.:0.1175   1st Qu.:0.08132
Median :0.03311   Median :0.1586   Median :0.13667
Mean   :0.06264   Mean   :0.1811   Mean   :0.17444
3rd Qu.:0.07170   3rd Qu.:0.2187   3rd Qu.:0.22680
Max.   :1.00000   Max.   :1.0000   Max.   :1.00000
 concavity_se     concave.points_se  symmetry_se
Min.   :0.00000   Min.   :0.0000    Min.   :0.0000
1st Qu.:0.03811   1st Qu.:0.1447    1st Qu.:0.1024
Median :0.06538   Median :0.2070    Median :0.1526
Mean   :0.08054   Mean   :0.2235    Mean   :0.1781
3rd Qu.:0.10619   3rd Qu.:0.2787    3rd Qu.:0.2195
Max.   :1.00000   Max.   :1.0000    Max.   :1.0000
fractal_dimension_se  radius_worst    texture_worst
Min.   :0.00000      Min.   :0.0000   Min.   :0.0000
1st Qu.:0.04675      1st Qu.:0.1807   1st Qu.:0.2415
Median :0.07919      Median :0.2504   Median :0.3569
Mean   :0.10019      Mean   :0.2967   Mean   :0.3640
3rd Qu.:0.12656      3rd Qu.:0.3863   3rd Qu.:0.4717
Max.   :1.00000      Max.   :1.0000   Max.   :1.0000
perimeter_worst    area_worst      smoothness_worst
Min.   :0.0000   Min.   :0.00000   Min.   :0.0000
1st Qu.:0.1678   1st Qu.:0.08113   1st Qu.:0.3000
Median :0.2353   Median :0.12321   Median :0.3971
Mean   :0.2831   Mean   :0.17091   Mean   :0.4041
3rd Qu.:0.3735   3rd Qu.:0.22090   3rd Qu.:0.4942
Max.   :1.0000   Max.   :1.00000   Max.   :1.0000
compactness_worst concavity_worst   concave.points_worst
Min.   :0.0000    Min.   :0.00000   Min.   :0.0000
1st Qu.:0.1163    1st Qu.:0.09145   1st Qu.:0.2231
Median :0.1791    Median :0.18107   Median :0.3434
Mean   :0.2202    Mean   :0.21740   Mean   :0.3938
3rd Qu.:0.3025    3rd Qu.:0.30583   3rd Qu.:0.5546
Max.   :1.0000    Max.   :1.00000   Max.   :1.0000
symmetry_worst   fractal_dimension_worst
Min.   :0.0000   Min.   :0.0000
1st Qu.:0.1851   1st Qu.:0.1077
Median :0.2478   Median :0.1640
Mean   :0.2633   Mean   :0.1896
3rd Qu.:0.3182   3rd Qu.:0.2429
Max.   :1.0000   Max.   :1.0000
```

- Ahora sí están en la misma escala. Todas están entre 0 y 1.

5. Bases de entrenamiento y prueba

- Bajo el método kNN una fracción de los datos se usa para entrenar el modelo, y el resto para probarlos.
- Tenemos 569 observaciones. Usaremos las primeras 469 para entrenar el modelo y lo pondremos a prueba con las restantes 100.
- Creemos la base de entrenamiento:

Hide

Hide

```
wbcd_train <- wbcd_n[1:469, ]
```

- Y la de prueba:

Hide

Hide

```
wbcd_test <- wbcd_n[470:569, ]
```

- Recordemos que `wbcd_n` excluye nuestra variable de interés, `diagnosis` . Debemos dividir las observaciones de esta variable para nuestra base de entrenamiento y la de prueba:

Hide

Hide

```
wbcd_train_diagnosis <- wbcd[1:469, 1]
wbcd_test_diagnosis <- wbcd[470:569, 1]
```

- Así, hemos creado dos vectores con los diagnósticos para los ejemplos en la base de entrenamiento y prueba.

6. Entrenamiento del Modelo

- En realidad bajo kNN no tenemos un modelo paramétrico.
- Por eso lo llamamos aprendizaje perezoso.
- Lo único que debemos hacer es almacenar los datos de insumo de manera estructurada.
- Para hacer este procedimiento, usamos el paquete `class`. Este paquete se compone de una serie de funciones para hacer clasificación.

Hide

Hide

```
install.packages("class")
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed

  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
  0     0    0     0    0     0      0      0 --:--:--  0:00:01 --:--:--     0
100 86276  100 86276    0     0  40266      0  0:00:02  0:00:02 --:--:-- 40278
tar: Failed to set default locale
```

```

The downloaded binary packages are in
    /var/folders/cw/_9zpljws58jbbv49mcb86l6c0000gn/T//Rtmp4wL8Gz/downloaded_packages
```

- Y lo cargamos:

Hide

Hide

```
library(class)
```

- Dentro del paquete `class`, la función `knn()` nos permite implementar el algortimo kNN.
- Para cada caso en los datos de prueba, se identifican los k vecinos más cercanos del conjunto de entrenamiento. Estos se hace calculando las distancias euclídeas.

- La clase asignada a cada caso es la que correspona a la mayoría de los k vecinos. En caso de empate, se determina aleatoriamente.
- En la función debemos especificar la base de entrenamiento, de prueba, el vector de clase y el número de vecinos k. La sintaxis es de la forma:

```
p <- knn(train, test, class, k)
```

- En nuestro caso, ya definimos las bases de entrenamiento y prueba, y el vector de clase. Falta definir k.
- Si usamos la heurística más utilizada: $k = \sqrt{(469)} \approx 21$. Aproximamos a 21 para tener un número impar, y reducir la probabilidad de empate.
- Con esto, podemos implementar la función:

Hide

Hide

```
wbcd_test_pred <- knn(train=wbcd_train, test=wbcd_test, cl=wbcd_train_diagnosis, k=21
)
```

- La función knn() crea un vector de predicciones de la clase a la que pertence cada caso en el conjunto de prueba:

Hide

Hide

```
wbcd_test_pred
```

```
  [1] Benign Benign Benign Benign Benign Benign Benign Benign Benign
 [10] Benign Malign Benign Benign Benign Benign Benign Benign Benign
 [19] Malign Benign Benign Benign Benign Malign Benign Benign Benign
 [28] Benign Benign Malign Malign Benign Malign Benign Malign Benign
 [37] Benign Benign Benign Benign Malign Benign Benign Malign Benign
 [46] Benign Benign Malign Malign Benign Benign Benign Malign Benign
 [55] Benign Benign Benign Benign Benign Benign Benign Benign Benign
 [64] Benign Malign Benign Malign Malign Benign Benign Benign Benign
 [73] Benign Benign Benign Benign Benign Benign Benign Benign Benign
 [82] Benign Benign Benign Benign Benign Benign Benign Benign Benign
 [91] Benign Benign Benign Malign Malign Malign Malign Malign Malign
[100] Benign
Levels: Benign Malign
```

7. Evaluación del Desempeño del Modelo

- ¿Qué tan preciso es nuestro modelo?
- La ventaja de lo que acabamos de hacer es que podemos comparar las predicciones del modelo, contenidas en el vector `wbcd_test_pred`, con la clasificación real de cada caso, en el vector `wbcd_test_diagnosis`.
- Para hacer esta comparación, usaremos el paquete `gmodels`. Este paquete contiene la función `CrossTable()`, que nos permite tabular los dos vectores.

Hide

Hide

```
install.packages("gmodels")
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 71885  100 71885    0     0   120k      0 --:--:-- --:--:-- --:--:--  120k
tar: Failed to set default locale
```

```

The downloaded binary packages are in
    /var/folders/cw/_9zpljws58jbbv49mcb86l6c0000gn/T//Rtmp4wL8Gz/downloaded_packages
```

Hide

Hide

```
library(gmodels)
```

- Y usamos la función `CrossTable()` para tabular. Desactivamos el parámetro `chisq`, porque no nos interesa el estadístico chi cuadrado que arroja la tabla.

Hide

Hide

```
CrossTable(x=wbcd_test_diagnosis, y=wbcd_test_pred, prop.chisq=FALSE)
```

```
   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  100


                     | wbcd_test_pred
wbcd_test_diagnosis |    Benign |    Malign | Row Total |
--------------------|-----------|-----------|-----------|
             Benign |        77 |         0 |        77 |
                    |     1.000 |     0.000 |     0.770 |
                    |     0.975 |     0.000 |           |
                    |     0.770 |     0.000 |           |
--------------------|-----------|-----------|-----------|
             Malign |         2 |        21 |        23 |
                    |     0.087 |     0.913 |     0.230 |
                    |     0.025 |     1.000 |           |
                    |     0.020 |     0.210 |           |
--------------------|-----------|-----------|-----------|
       Column Total |        79 |        21 |       100 |
                    |     0.790 |     0.210 |           |
--------------------|-----------|-----------|-----------|
```

- Nótese que de 77 casos realmente benignos, los 77 fueron correctamente clasificados como benignos por el modelo. Estos son los “true negatives”, o verdaderos negativos.
- Por su parte, hay 23 casos realmente malignos, de los cuales 21 casos son acertadamente clasificados por el algoritmo. Estos son los “true positives”, o verdaderos positivos
- En este caso tenemos 0 falsos positivos. No hay ningún caso realmente benigno que sea clasificado erróneamente como maligno.
- Sin embargo, hay 2 falsos negativos. Casos realmente malignos que son clasificados erróneamente como benignos.

- En esta apliacación los falsos negativos son graves: pueden llevar a no tratar dos casos que deberían ser tratados porque en realidad son malignos. El algo grave para esos pacientes.
- Los falsos positivos (que no tenemos en este caso) también son graves. Llevan a desperdiciar recursos tratando casos que no son cancerosos.
- En el agregado, nuestro modelo se equivoca en 2 de 100 casos. Tiene una precisión del 98%. Nada mal.

8. Robustez: estandarizacion

- En el ejercicio anterior rescalamos las variables usando la normalización.
- El problema de este método es que se elimina el efecto de los outliers.
- En una situación como la de tumores, los outliers pueden ser cruciales.
- Por tanto, repliquemos el proceso anterior, pero estandarizando en lugar de normalizando.
- La función `scale()` rescala valores estandarizándolos, es decir, calculando los z-scores. Además, se aplica directamente a todo el data frame, luego no es necesario usar la función `lapply()` en este caso.

Hide

Hide

```
wbcd_z <- as.data.frame(scale(wbcd[-1]))
```

- Podemos verificar el proceso:

Hide

Hide

```
summary(wbcd_z)
```

```
  radius_mean       texture_mean     perimeter_mean
 Min.   :-2.0279   Min.   :-2.2273   Min.   :-1.9828
 1st Qu.:-0.6888   1st Qu.:-0.7253   1st Qu.:-0.6913
 Median :-0.2149   Median :-0.1045   Median :-0.2358
 Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000
 3rd Qu.: 0.4690   3rd Qu.: 0.5837   3rd Qu.: 0.4992
 Max.   : 3.9678   Max.   : 4.6478   Max.   : 3.9726
```

```
  area_mean       smoothness_mean    compactness_mean
Min.   :-1.4532   Min.   :-3.10935   Min.   :-1.6087
1st Qu.:-0.6666   1st Qu.:-0.71034   1st Qu.:-0.7464
Median :-0.2949   Median :-0.03486   Median :-0.2217
Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.0000
3rd Qu.: 0.3632   3rd Qu.: 0.63564   3rd Qu.: 0.4934
Max.   : 5.2459   Max.   : 4.76672   Max.   : 4.5644
concavity_mean    concave.points_mean symmetry_mean
Min.   :-1.1139   Min.   :-1.2607     Min.   :-2.74171
1st Qu.:-0.7431   1st Qu.:-0.7373     1st Qu.:-0.70262
Median :-0.3419   Median :-0.3974     Median :-0.07156
Mean   : 0.0000   Mean   : 0.0000     Mean   : 0.00000
3rd Qu.: 0.5256   3rd Qu.: 0.6464     3rd Qu.: 0.53031
Max.   : 4.2399   Max.   : 3.9245     Max.   : 4.48081
fractal_dimension_mean   radius_se         texture_se
Min.   :-1.8183        Min.   :-1.0590   Min.   :-1.5529
1st Qu.:-0.7220        1st Qu.:-0.6230   1st Qu.:-0.6942
Median :-0.1781        Median :-0.2920   Median :-0.1973
Mean   : 0.0000        Mean   : 0.0000   Mean   : 0.0000
3rd Qu.: 0.4706        3rd Qu.: 0.2659   3rd Qu.: 0.4661
Max.   : 4.9066        Max.   : 8.8991   Max.   : 6.6494
 perimeter_se        area_se         smoothness_se
Min.   :-1.0431   Min.   :-0.7372   Min.   :-1.7745
1st Qu.:-0.6232   1st Qu.:-0.4943   1st Qu.:-0.6235
Median :-0.2864   Median :-0.3475   Median :-0.2201
Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000
3rd Qu.: 0.2428   3rd Qu.: 0.1067   3rd Qu.: 0.3680
Max.   : 9.4537   Max.   :11.0321   Max.   : 8.0229
compactness_se     concavity_se     concave.points_se
Min.   :-1.2970   Min.   :-1.0566   Min.   :-1.9118
1st Qu.:-0.6923   1st Qu.:-0.5567   1st Qu.:-0.6739
Median :-0.2808   Median :-0.1989   Median :-0.1404
Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000
3rd Qu.: 0.3893   3rd Qu.: 0.3365   3rd Qu.: 0.4722
Max.   : 6.1381   Max.   :12.0621   Max.   : 6.6438
 symmetry_se       fractal_dimension_se  radius_worst
Min.   :-1.5315   Min.   :-1.0960      Min.   :-1.7254
1st Qu.:-0.6511   1st Qu.:-0.5846      1st Qu.:-0.6743
Median :-0.2192   Median :-0.2297      Median :-0.2688
Mean   : 0.0000   Mean   : 0.0000      Mean   : 0.0000
3rd Qu.: 0.3554   3rd Qu.: 0.2884      3rd Qu.: 0.5216
Max.   : 7.0657   Max.   : 9.8429      Max.   : 4.0906
texture_worst      perimeter_worst     area_worst
Min.   :-2.22204   Min.   :-1.6919   Min.   :-1.2213
1st Qu.:-0.74797   1st Qu.:-0.6890   1st Qu.:-0.6416
Median :-0.04348   Median :-0.2857   Median :-0.3409
Mean   : 0.00000   Mean   : 0.0000   Mean   : 0.0000
3rd Qu.: 0.65776   3rd Qu.: 0.5398   3rd Qu.: 0.3573
Max.   : 3.88249   Max.   : 4.2836   Max.   : 5.9250
smoothness_worst  compactness_worst concavity_worst
```

```
 Min.   :-2.6803   Min.   :-1.4426   Min.   :-1.3047
 1st Qu.:-0.6906   1st Qu.:-0.6805   1st Qu.:-0.7558
 Median :-0.0468   Median :-0.2693   Median :-0.2180
 Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000
 3rd Qu.: 0.5970   3rd Qu.: 0.5392   3rd Qu.: 0.5307
 Max.   : 3.9519   Max.   : 5.1084   Max.   : 4.6965
 concave.points_worst symmetry_worst    fractal_dimension_worst
 Min.   :-1.7435      Min.   :-2.1591   Min.   :-1.6004
 1st Qu.:-0.7557      1st Qu.:-0.6413   1st Qu.:-0.6913
 Median :-0.2233      Median :-0.1273   Median :-0.2163
 Mean   : 0.0000      Mean   : 0.0000   Mean   : 0.0000
 3rd Qu.: 0.7119      3rd Qu.: 0.4497   3rd Qu.: 0.4504
 Max.   : 2.6835      Max.   : 6.0407   Max.   : 6.8408
```

- En efecto la media es 0.
- Ahora replicamos el proceso anterior. Dividimos la base en entrenamiento y prueba:

Hide

Hide

```
wbcd_train <- wbcd_z[1:469, ]
wbcd_test <- wbcd_z[470:569, ]
```

Y los vectores del outcome de interés:

Hide

Hide

```
wbcd_train_diagnosis <- wbcd[1:469, 1]
wbcd_test_diagnosis <- wbcd[470:569, 1]
```

- Y corremos el modelo con la funci??n `knn()` :

Hide

Hide

```
wbcd_test_pred<- knn(train=wbcd_train, test=wbcd_test, cl=wbcd_train_diagnosis, k=21)
```

- Podemos ver el vector de predicciones:

Hide

Hide

```
wbcd_test_pred
```

```
  [1] Benign Benign Benign Benign Benign Benign Benign Benign Benign
 [10] Benign Malign Benign Benign Benign Benign Benign Benign Benign
 [19] Malign Benign Benign Benign Benign Malign Benign Benign Benign
 [28] Benign Benign Malign Malign Benign Malign Benign Malign Benign
 [37] Benign Benign Benign Benign Malign Benign Benign Malign Benign
 [46] Benign Benign Malign Malign Benign Benign Benign Malign Benign
 [55] Benign Benign Benign Benign Benign Benign Benign Benign Benign
 [64] Benign Malign Benign Malign Malign Benign Benign Benign Benign
 [73] Benign Benign Benign Benign Benign Benign Benign Benign Benign
 [82] Benign Benign Benign Benign Benign Benign Benign Benign Benign
 [91] Benign Benign Benign Malign Malign Malign Malign Malign Malign
[100] Benign
Levels: Benign Malign
```

- Y creamos las tablas para ver la precisión del modelo:

Hide

Hide

```
CrossTable(x=wbcd_test_diagnosis, y=wbcd_test_pred, prop.chisq=FALSE)
```

```
   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  100


                     | wbcd_test_pred
wbcd_test_diagnosis |    Benign |    Malign | Row Total |
--------------------|-----------|-----------|-----------|
             Benign |        77 |         0 |        77 |
                    |     1.000 |     0.000 |     0.770 |
                    |     0.975 |     0.000 |           |
                    |     0.770 |     0.000 |           |
--------------------|-----------|-----------|-----------|
             Malign |         2 |        21 |        23 |
                    |     0.087 |     0.913 |     0.230 |
                    |     0.025 |     1.000 |           |
                    |     0.020 |     0.210 |           |
--------------------|-----------|-----------|-----------|
       Column Total |        79 |        21 |       100 |
                    |     0.790 |     0.210 |           |
--------------------|-----------|-----------|-----------|
```

- Vemos que la precisión del modelo es idéntica en este caso.
- 2 de 100 casos son clasificados incorrectamente como falsos negativos. Es decir, seguimos con una precisión del 98%.