

Machine Learning

00R Team

Abril 2016

Índice

1. Introducción	1
2. Algoritmo K Nearest Neighbors (KNN)	1
3. Máquina de soporte de vectores (SVM)	4
4. Clasificador Naïve Bayes	8
5. Redes Neuronales	9
6. Clasificación no supervisada. K-means como clasificador-predicador	16
7. Evaluar la eficacia del modelo	18

1. Introducción

El aprendizaje automatizado —*machine learning (ML)*— es una rama de la inteligencia artificial cuyo objetivo es que una máquina aprenda a partir de la experiencia. Básicamente los algoritmos toman un conjunto de datos, los analizan para buscar en ellos ciertas pautas y una vez identificadas, las emplean para realizar predicciones. En otras palabras se trata de predecir el comportamiento futuro a partir de comportamientos pasados observados.

Dependiendo de cómo se aborde el problema del ML, los diferentes algoritmos se pueden agrupar en:

- Aprendizaje supervisado

Cuando las entradas al sistema (la base de conocimiento) están formadas por un conjunto de datos etiquetados a priori. Es decir sabemos la clasificación correcta de un conjunto de datos, y a partir de ellos se va a generar una función predictora.

- Aprendizaje no supervisado

Las entradas al sistema están formadas por un conjunto de datos de los que se desconoce su clasificación correcta. En este caso, se espera que el algoritmo sea capaz de reconocer patrones para poder etiquetar y clasificar nuevos datos de entrada.

2. Algoritmo K Nearest Neighbors (KNN)

Algoritmo de clasificación supervisada donde dado un objeto a clasificar y sus K vecinos más cercanos, será clasificado al grupo con mayor probabilidad de pertenencia.

Ejemplo en R

```
data( iris )
```

Preprocesar datos:

Incluye normalizar si es necesario, reducir el número de variables si fuese necesario...

P.e. $\frac{x-\bar{x}}{\sigma^2}$, es lo que hace la función `scale()`

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

```
NormIris <- as.data.frame( lapply( iris[ , 1:4 ], normalize ) )  
summary( NormIris )
```

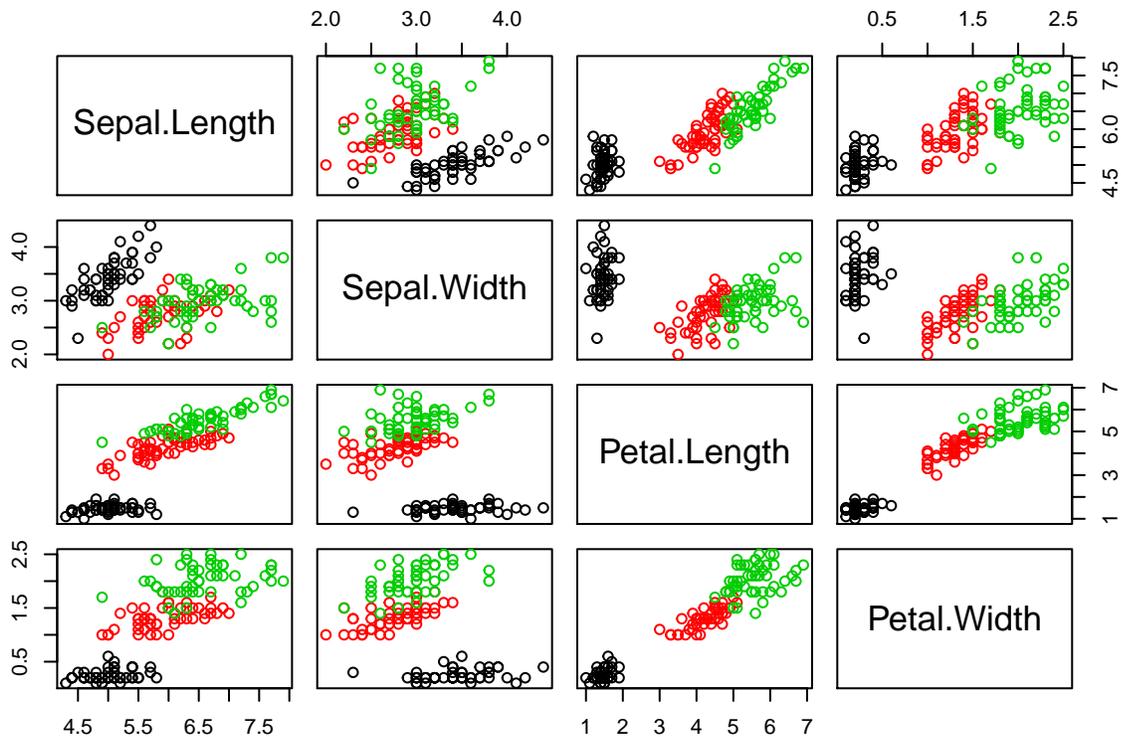
```
## Sepal.Length Sepal.Width Petal.Length Petal.Width  
## Min. :0.0000 Min. :0.0000 Min. :0.0000 Min. :0.00000  
## 1st Qu.:0.2222 1st Qu.:0.3333 1st Qu.:0.1017 1st Qu.:0.08333  
## Median :0.4167 Median :0.4167 Median :0.5678 Median :0.50000  
## Mean :0.4287 Mean :0.4406 Mean :0.4675 Mean :0.45806  
## 3rd Qu.:0.5833 3rd Qu.:0.5417 3rd Qu.:0.6949 3rd Qu.:0.70833  
## Max. :1.0000 Max. :1.0000 Max. :1.0000 Max. :1.00000
```

```
NormIris <- scale( iris[ , 1:4 ], scale = TRUE, center = TRUE )  
summary( NormIris )
```

```
## Sepal.Length Sepal.Width Petal.Length Petal.Width  
## Min. :-1.86378 Min. :-2.4258 Min. :-1.5623 Min. :-1.4422  
## 1st Qu.:-0.89767 1st Qu.:-0.5904 1st Qu.:-1.2225 1st Qu.:-1.1799  
## Median :-0.05233 Median :-0.1315 Median : 0.3354 Median : 0.1321  
## Mean : 0.00000 Mean : 0.0000 Mean : 0.0000 Mean : 0.0000  
## 3rd Qu.: 0.67225 3rd Qu.: 0.5567 3rd Qu.: 0.7602 3rd Qu.: 0.7880  
## Max. : 2.48370 Max. : 3.0805 Max. : 1.7799 Max. : 1.7064
```

Previsualizar datos

```
plot( iris[ , 1:4 ], col = iris$Species )
```



Generar las muestras

```
#generar muestras
iris.train <- iris[ sample( c( 1:150 ), 100 ), 1:5 ]

iris.test <- iris[ sample( c( 1:150 ), 50 ), 1:5 ]
```

Entrenar/testar clasificador

```
# Genera modelo y hace predicción a la vez
iris.pred <- knn( train = iris.train[ , 1:4 ], test = iris.test[ , 1:4 ],
                 cl = iris.train[ , 5 ], k = 3 )
```

```
# predicción
iris.pred
```

```
## [1] virginica virginica setosa versicolor setosa setosa
## [7] versicolor setosa versicolor setosa virginica virginica
## [13] versicolor setosa virginica versicolor versicolor versicolor
## [19] setosa virginica setosa setosa virginica virginica
## [25] virginica setosa setosa versicolor versicolor virginica
## [31] setosa versicolor virginica versicolor versicolor setosa
## [37] versicolor versicolor setosa setosa setosa virginica
## [43] versicolor virginica virginica setosa setosa setosa
## [49] virginica versicolor
## Levels: setosa versicolor virginica
```

```
# realidad
iris.test[ , 5 ]
```

```
## [1] virginica virginica setosa versicolor setosa setosa
## [7] versicolor setosa versicolor setosa virginica virginica
## [13] versicolor setosa virginica versicolor versicolor versicolor
```

```
## [19] setosa      virginica setosa      setosa      virginica virginica
## [25] virginica setosa      setosa      versicolor versicolor versicolor
## [31] setosa      versicolor virginica versicolor versicolor setosa
## [37] virginica versicolor setosa      setosa      setosa      virginica
## [43] versicolor virginica virginica setosa      setosa      setosa
## [49] virginica versicolor
## Levels: setosa versicolor virginica
```

Validar resultados

```
table( Predic = iris.pred, Test = iris.test[ , 5 ] )
```

```
##           Test
## Predic   setosa versicolor virginica
## setosa      19         0         0
## versicolor   0         15        1
## virginica    0         1        14
```

Si encuentro un iris en el campo y mido pétalo y sépalo, puedo utilizar mi clasificador para determinar a que variedad pertenece.

```
miIris <- c( 5.0, 3.5, 1.3, 0.1 )
mi_predict <- knn( train = iris.train[ , 1:4 ], test = miIris,
                  cl = iris.train[ , 5 ], k = 3, prob = TRUE )
```

```
mi_predict
```

```
## [1] setosa
## attr(,"prob")
## [1] 1
## Levels: setosa versicolor virginica
```

3. Máquina de soporte de vectores (SVM)

Algoritmo de clasificación supervisada, donde dado un conjunto de datos etiquetados, el algoritmo construye un límite o frontera óptimo, llamado hiperplano, que separa los datos según su categoría para posteriormente asignar nuevos datos al grupo con mayor probabilidad de pertenencia.

Ejemplo en R

```
data( iris )
```

Preprocesar datos:

Incluye normalizar si es necesario, reducir el número de variables si fuese necesario...

Generar las muestras

```
#generar muestras
iris.train <- iris[ sample( c( 1:150 ), 100 ), 1:5 ]

iris.test <- iris[ sample( c( 1:150 ), 30 ), 1:5 ]
```

Aplicar SVM

```
# Esta función sólo genera el modelo
svm_model <- svm( Species ~ . , data = iris.train, scale = TRUE, kernel = "radial", probability = TRUE )
summary( svm_model )
```

```
##
## Call:
## svm(formula = Species ~ ., data = iris.train, kernel = "radial",
##      probability = TRUE, scale = TRUE)
##
##
## Parameters:
##   SVM-Type:  C-classification
##   SVM-Kernel: radial
##     cost: 1
##   gamma: 0.25
##
## Number of Support Vectors: 44
##
## ( 8 17 19 )
##
##
## Number of Classes: 3
##
## Levels:
## setosa versicolor virginica
```

Hacer predicción con el conjunto de datos de iris.test.

```
SVM_predict <- predict( svm_model, iris.test[,1:4], probability=TRUE, decision.values=TRUE )
SVM_predict
```

```
##      91      143      52      119      43      81
## versicolor virginica versicolor virginica setosa versicolor
##      49      40      10      86      78      29
## setosa setosa setosa versicolor virginica setosa
##      63      64      72      118      127      148
## versicolor versicolor versicolor virginica virginica virginica
##      85      44      55      95      15      17
## versicolor setosa versicolor versicolor setosa setosa
##      31      54      1      18      35      116
## setosa versicolor setosa setosa setosa virginica
## attr(,"decision.values")
## setosa/virginica setosa/versicolor virginica/versicolor
## 91 -0.8011587 -1.3539253 -0.9623278
## 143 -1.1668284 -1.1741559 0.9334494
## 52 -0.7324376 -1.1675859 -1.0003317
## 119 -0.9999640 -0.1950084 1.3018699
## 43 1.0053878 1.1241182 -0.3372870
## 81 -0.5946436 -1.2865520 -1.2811702
## 49 1.0917048 1.1870994 -0.4630378
## 40 1.0737397 1.2060808 -0.5994227
## 10 1.0534240 1.1419490 -0.5921337
## 86 -0.5439425 -0.9998309 -1.0000462
## 78 -1.0193973 -1.1156730 0.2033642
## 29 1.0647067 1.2008190 -0.6153962
## 63 -0.6068452 -1.0443882 -1.2187311
## 64 -0.8783982 -1.2960340 -0.7115705
## 72 -0.7243465 -1.2648358 -1.3706251
## 118 -0.9998538 -0.2262171 0.9997948
```

```
## 127      -1.0935455      -1.2234191          0.4338866
## 148      -1.0947090      -1.0503445          1.0028150
## 85       -0.7295351      -1.1646988         -0.6521537
## 44       1.0234286       1.0619228         -0.7098300
## 55      -0.9491372      -1.2007362         -0.4249463
## 95      -0.7924793      -1.3557290         -1.0424277
## 15       1.0246431       1.0600813         -0.2841489
## 17       1.1088278       1.1461736         -0.3792406
## 31       1.0541487       1.0992015         -0.6249102
## 54      -0.7948938      -1.3278935         -0.8954043
## 1        1.0881427       1.2276465         -0.5112211
## 18       1.0918244       1.2110172         -0.5637262
## 35       1.0586751       1.1128493         -0.6538730
## 116     -0.9900481      -0.8774992          1.2019659
```

```
## attr(,"probabilities")
##      setosa  virginica versicolor
## 91  0.007701210 0.04598016 0.94631863
## 143 0.007431326 0.95623657 0.03633210
## 52  0.013462977 0.04149673 0.94504029
## 119 0.021150252 0.96499039 0.01385936
## 43  0.951333654 0.02332924 0.02533711
## 81  0.009724830 0.01745815 0.97281702
## 49  0.963766649 0.01586458 0.02036877
## 40  0.966507921 0.01419212 0.01929996
## 10  0.960474144 0.01643187 0.02309398
## 86  0.025308380 0.04230590 0.93238572
## 78  0.013081347 0.69047238 0.29644627
## 29  0.965835729 0.01454687 0.01961740
## 63  0.020612398 0.02225421 0.95713340
## 64  0.010451721 0.09850416 0.89104412
## 72  0.009979052 0.01299062 0.97703033
## 118 0.022292243 0.94757493 0.03013283
## 127 0.009136200 0.82613150 0.16473230
## 148 0.009799968 0.96095434 0.02924569
## 85  0.015381441 0.11735630 0.86726226
## 44  0.954062741 0.01626968 0.02966758
## 55  0.012442447 0.22097849 0.76657906
## 95  0.007488067 0.03581273 0.95669920
## 15  0.946579070 0.02372227 0.02969866
## 17  0.960753612 0.01645033 0.02279606
## 31  0.957756762 0.01620872 0.02603452
## 54  0.008553078 0.05660852 0.93483840
## 1    0.966618013 0.01528039 0.01810160
## 18  0.966583583 0.01461942 0.01879700
## 35  0.959433102 0.01564938 0.02491752
## 116 0.014589621 0.96933781 0.01607257
## Levels: setosa versicolor virginica
```

Contrastar la predicción con la realidad

```
# Versión simple
table( SVM_predict, Realidad = as.factor( iris.test[ , 5 ] ) )
```

```
##              Realidad
## SVM_predict  setosa versicolor virginica
```

```
## setosa      12      0      0
## versicolor  0      11     0
## virginica   0      1      6
```

```
# Versión más completa
CrossTable( x = SVM_predict, y = iris.test[ , 5 ], dnn = c( "Predicción", "Real" ) )
```

```
##
##
## Cell Contents
## |-----|
## |                N |
## | Chi-square contribution |
## |      N / Row Total |
## |      N / Col Total |
## |      N / Table Total |
## |-----|
##
##
## Total Observations in Table: 30
##
##
##      | Real
## Predicción | setosa | versicolor | virginica | Row Total |
## -----|-----|-----|-----|-----|
## setosa |      12 |          0 |          0 |          12 |
##      | 10.800 |      4.800 |      2.400 |          |
##      |  1.000 |      0.000 |      0.000 |      0.400 |
##      |  1.000 |      0.000 |      0.000 |          |
##      |  0.400 |      0.000 |      0.000 |          |
## -----|-----|-----|-----|
## versicolor |          0 |          11 |          0 |          11 |
##      |  4.400 |      9.900 |      2.200 |          |
##      |  0.000 |      1.000 |      0.000 |      0.367 |
##      |  0.000 |      0.917 |      0.000 |          |
##      |  0.000 |      0.367 |      0.000 |          |
## -----|-----|-----|-----|
## virginica |          0 |          1 |          6 |          7 |
##      |  2.800 |      1.157 |     15.114 |          |
##      |  0.000 |      0.143 |      0.857 |      0.233 |
##      |  0.000 |      0.083 |      1.000 |          |
##      |  0.000 |      0.033 |      0.200 |          |
## -----|-----|-----|-----|
## Column Total |          12 |          12 |          6 |          30 |
##      |  0.400 |      0.400 |      0.200 |          |
## -----|-----|-----|-----|
##
##
```

Y con mi muestra encontrada en el campo

```
miIris <- matrix( c( 5.0, 3.5, 1.3, 0.1 ), nrow = 1, ncol = 4 )
colnames( miIris ) <- colnames( iris[ , 1:4 ] )

# predict requiere un matrix o data.frame no un vector
predict( svm_model, miIris )
```

```
##      1
## setosa
## Levels: setosa versicolor virginica
```

4. Clasificador Naïve Bayes

Es un clasificador probabilístico “ingenuo” basado en el teorema de Bayes, que asume que la probabilidad de cada variable es independiente de las demás. Básicamente su lógica se basa en que dado un conjunto de datos de entrenamiento etiquetados, el clasificador calcula la probabilidad observada para cada clase, en función de los valores de sus variables. Cuando es usado posteriormente para predecir datos sin etiquetar, asigna estos datos a la clase con mayor probabilidad de pertenencia.

```
iris.NB <- naiveBayes( Species ~ ., data = iris.train )
iris.NB
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##      setosa versicolor  virginica
##      0.33      0.33      0.34
##
## Conditional probabilities:
##      Sepal.Length
## Y      [,1]      [,2]
## setosa  5.024242 0.3725170
## versicolor 5.933333 0.5265849
## virginica 6.605882 0.6591736
##
##      Sepal.Width
## Y      [,1]      [,2]
## setosa  3.445455 0.4228556
## versicolor 2.784848 0.3392282
## virginica 2.914706 0.2986153
##
##      Petal.Length
## Y      [,1]      [,2]
## setosa  1.442424 0.1936981
## versicolor 4.239394 0.5005868
## virginica 5.550000 0.5674237
##
##      Petal.Width
## Y      [,1]      [,2]
## setosa  0.2333333 0.09574271
## versicolor 1.3454545 0.20629304
## virginica 1.9764706 0.27309541
```

```
predNB <- predict( iris.NB, iris.test[ , 1:4 ] )
predNB
```

```
## [1] versicolor virginica versicolor virginica setosa versicolor
## [7] setosa setosa setosa versicolor virginica setosa
## [13] versicolor versicolor versicolor virginica virginica virginica
## [19] versicolor setosa versicolor versicolor setosa setosa
## [25] setosa versicolor setosa setosa setosa virginica
## Levels: setosa versicolor virginica
```

Contrastar la predicción

```
table( Predicción = predNB, Real = iris.test[ , 5 ] )
```

```
##           Real
## Predicción setosa versicolor virginica
## setosa      12         0         0
## versicolor  0         11        0
## virginica   0         1         6
```

Y con mi muestra encontrada en el campo

```
miIris<-matrix( c( 5.0, 3.5, 1.3, 0.1 ), nrow = 1, ncol = 4 )
colnames( miIris ) <-colnames( iris[ , 1:4 ] )

# predict requiere un matrix o data.frame no un vector
predict( iris.NB, miIris )
```

```
## [1] setosa
## Levels: setosa versicolor virginica
```

5. Redes Neuronales

Es un modelo de aprendizaje automatizado inspirados en los modelos neuronales biológicos. Constituidos por una serie de nodos (neuronas) y una serie de conexiones entre los nodos (sinapsis).

La topología básica consta de unos nodos de entrada al sistema, generalmente tantos nodos como variables. Una serie de capas ocultas intermedias con un número variable de nodos, y unos nodos de salida, uno por cada respuesta. Cada nodo de entrada tiene asociado un peso W_i y en cada nodo se aplica una función de activación (opcional) y una de propagación con la suma de las entradas ponderadas por sus pesos.

Aunque hay muchas topologías, ésta es conocida como *perceptrón*

5.0.1. Ejemplo de clasificación mediante NN

Preprocesar datos:

Incluye normalizar si es necesario, reducir el número de variables si fuese necesario. . . .

Generar las muestras

```
#generar muestras
iris.train <- iris[ sample( c( 1:150 ), 100 ), 1:5 ]
iris.test <- iris[ sample( c( 1:150 ), 50 ), 1:5 ]
```

Binarizar las categorías (*dummy transformation*)

```

iris.train <- cbind( iris.train, iris.train$Species == "setosa" )
iris.train <- cbind( iris.train, iris.train$Species == "versicolor" )
iris.train <- cbind( iris.train, iris.train$Species == "virginica" )

names( iris.train )[ 6 ] <- "setosa"
names( iris.train )[ 7 ] <- "versicolor"
names( iris.train )[ 8 ] <- "virginica"

```

Entrenar la red neuronal

```

iris.nnt <- neuralnet( setosa + versicolor + virginica ~ Sepal.Length +
  Sepal.Width +
  Petal.Length +
  Petal.Width,
  data = iris.train, hidden = c( 3, 2 ) )
plot( iris.nnt, col.intercept = "blue" )

```

Predicción

```

pred.nn <- compute( iris.nnt, iris.test[ 1:4 ] )
pred.nn

```

```

## $neurons
## $neurons[[1]]
##      1 Sepal.Length Sepal.Width Petal.Length Petal.Width
## 143 1          5.8          2.7          5.1          1.9
## 26  1          5.0          3.0          1.6          0.2
## 85  1          5.4          3.0          4.5          1.5
## 83  1          5.8          2.7          3.9          1.2
## 93  1          5.8          2.6          4.0          1.2
## 114 1          5.7          2.5          5.0          2.0
## 88  1          6.3          2.3          4.4          1.3
## 78  1          6.7          3.0          5.0          1.7
## 79  1          6.0          2.9          4.5          1.5
## 76  1          6.6          3.0          4.4          1.4
## 82  1          5.5          2.4          3.7          1.0
## 107 1          4.9          2.5          4.5          1.7
## 136 1          7.7          3.0          6.1          2.3
## 31  1          4.8          3.1          1.6          0.2
## 90  1          5.5          2.5          4.0          1.3
## 47  1          5.1          3.8          1.6          0.2
## 58  1          4.9          2.4          3.3          1.0
## 81  1          5.5          2.4          3.8          1.1
## 44  1          5.0          3.5          1.6          0.6
## 141 1          6.7          3.1          5.6          2.4
## 33  1          5.2          4.1          1.5          0.1
## 61  1          5.0          2.0          3.5          1.0
## 103 1          7.1          3.0          5.9          2.1
## 130 1          7.2          3.0          5.8          1.6
## 120 1          6.0          2.2          5.0          1.5
## 116 1          6.4          3.2          5.3          2.3
## 128 1          6.1          3.0          4.9          1.8
## 126 1          7.2          3.2          6.0          1.8
## 95  1          5.6          2.7          4.2          1.3
## 65  1          5.6          2.9          3.6          1.3

```

```

## 1 1 5.1 3.5 1.4 0.2
## 22 1 5.1 3.7 1.5 0.4
## 111 1 6.5 3.2 5.1 2.0
## 77 1 6.8 2.8 4.8 1.4
## 3 1 4.7 3.2 1.3 0.2
## 50 1 5.0 3.3 1.4 0.2
## 28 1 5.2 3.5 1.5 0.2
## 80 1 5.7 2.6 3.5 1.0
## 4 1 4.6 3.1 1.5 0.2
## 118 1 7.7 3.8 6.7 2.2
## 27 1 5.0 3.4 1.6 0.4
## 51 1 7.0 3.2 4.7 1.4
## 96 1 5.7 3.0 4.2 1.2
## 106 1 7.6 3.0 6.6 2.1
## 101 1 6.3 3.3 6.0 2.5
## 84 1 6.0 2.7 5.1 1.6
## 105 1 6.5 3.0 5.8 2.2
## 5 1 5.0 3.6 1.4 0.2
## 89 1 5.6 3.0 4.1 1.3
## 129 1 6.4 2.8 5.6 2.1
##
## $neurons[[2]]
## [,1]
## 143 1
## 26 1
## 85 1
## 83 1
## 93 1
## 114 1
## 88 1
## 78 1
## 79 1
## 76 1
## 82 1
## 107 1
## 136 1
## 31 1
## 90 1
## 47 1
## 58 1
## 81 1
## 44 1
## 141 1
## 33 1
## 61 1
## 103 1
## 130 1
## 120 1
## 116 1
## 128 1
## 126 1
## 95 1
## 65 1
## 1 1

```



```

## 80 0.96083650768 0.0655952297374
## 4 0.99974609450 0.0038978498961
## 118 0.12082027074 0.0059235369590
## 27 0.99969510089 0.0024251761438
## 51 0.85099167951 0.0308157981374
## 96 0.90953620901 0.0131118148269
## 106 0.06496894031 0.0875878824594
## 101 0.04695865565 0.0104475900007
## 84 0.43990112180 0.0486976490251
## 105 0.09427865435 0.0350964812546
## 5 0.99989589924 0.0011074540068
## 89 0.89296620137 0.0130289231164
## 129 0.11641537312 0.0637438827167
##
## $neurons[[3]]
##      [,1]      [,2]      [,3]
## 143 1 0.10317595427 1.000000000e+00
## 26 1 0.95658774662 5.780314834e-144
## 85 1 0.11430860024 2.130270290e-22
## 83 1 0.11905696447 9.329889310e-37
## 93 1 0.11863857983 1.398579905e-35
## 114 1 0.10194747849 1.000000000e+00
## 88 1 0.11679663978 8.937213173e-31
## 78 1 0.11013309478 1.408463988e-09
## 79 1 0.11462929903 1.928516333e-23
## 76 1 0.11751495485 3.601606009e-32
## 82 1 0.12007887005 6.818174132e-40
## 107 1 0.10697050813 9.495373068e-01
## 136 1 0.10017769510 1.000000000e+00
## 31 1 0.95657178140 7.841414126e-144
## 90 1 0.11715806739 3.391242568e-31
## 47 1 0.95655698313 1.043298718e-143
## 58 1 0.12015001260 5.676495357e-40
## 81 1 0.11933576500 1.008356420e-37
## 44 1 0.95655830228 1.002570446e-143
## 141 1 0.09964063392 1.000000000e+00
## 33 1 0.95655620285 1.060096888e-143
## 61 1 0.12036382521 4.367123798e-41
## 103 1 0.10076558140 1.000000000e+00
## 130 1 0.10695487056 9.479942587e-01
## 120 1 0.10794506690 2.937941047e-03
## 116 1 0.10103668239 1.000000000e+00
## 128 1 0.10806806345 5.984682756e-03
## 126 1 0.10394950255 1.000000000e+00
## 95 1 0.11699143536 1.384950541e-30
## 65 1 0.11934757467 1.592971748e-37
## 1 1 0.95656174549 9.524379407e-144
## 22 1 0.95655783112 1.025016094e-143
## 111 1 0.10560686308 9.999982390e-01
## 77 1 0.11519683320 2.281782202e-25
## 3 1 0.95656732567 8.554004589e-144
## 50 1 0.95656716945 8.581978356e-144
## 28 1 0.95656222703 9.435181405e-144
## 80 1 0.12051841360 4.364883453e-41

```

```

## 4      1 0.95656840469 8.365945116e-144
## 118    1 0.10016184574 1.000000000e+00
## 27     1 0.95656219148 9.408774649e-144
## 51     1 0.11740570533 8.388657627e-32
## 96     1 0.11870212176 1.373638517e-35
## 106    1 0.09963657778 1.000000000e+00
## 101    1 0.09860717932 1.000000000e+00
## 84     1 0.10768744893 7.682806617e-02
## 105    1 0.09983118855 1.000000000e+00
## 5      1 0.95655909793 1.001977651e-143
## 89     1 0.11828441962 2.359002506e-34
## 129    1 0.10055260742 1.000000000e+00
##
##
## $net.result
##           [,1]           [,2]           [,3]
## 143  0.0007806842127 -0.00491424132405 1.00411996663665
## 26   0.9999979665369 -0.00006013805125 0.00005323139384
## 85  -0.0023745741133 1.00455513350833 -0.00214357061832
## 83   0.0032763195857 0.99889159637245 -0.00213118610643
## 93   0.0027784118795 0.99939061810858 -0.00213227732213
## 114 -0.0006812899893 -0.00344899606804 1.00411676257058
## 88   0.0005863712328 1.00158756327636 -0.00213708140355
## 78  -0.0073437247976 1.00953540086505 -0.00215445960335
## 79  -0.0019929195683 1.00417262504084 -0.00214273418333
## 76   0.0014412176373 1.00073080421928 -0.00213520792017
## 82   0.0044924604532 0.99767273448394 -0.00212852080925
## 107  0.0044686900088 0.04217046556439 0.95334963003667
## 136 -0.0027874590401 -0.00133811462718 1.00411214668555
## 31   0.9999789667850 -0.00004109578893 0.00005318975394
## 90   0.0010164959906 1.00115647614781 -0.00213613874113
## 47   0.9999613557771 -0.00002344537772 0.00005315115761
## 58   0.0045771251850 0.99758788032142 -0.00212833525783
## 81   0.0036081121954 0.99855906140244 -0.00213045894897
## 44   0.9999629256673 -0.00002501878045 0.00005315459819
## 141 -0.0034266003284 -0.00069754330972 1.00411074594200
## 33   0.9999604271947 -0.00002251471774 0.00005314912253
## 61   0.0048315775190 0.99733285866997 -0.00212777759950
## 103 -0.0020878322151 -0.00203930681299 1.00411367998934
## 130  0.0044247681685 0.04376726583369 0.95179683144321
## 120 -0.0098994410741 1.00914036596757 0.00079626049999
## 116 -0.0017652026199 -0.00236265826691 1.00411438706512
## 128 -0.0097030879098 1.00587761579598 0.00386249484886
## 126  0.0017012621072 -0.00583687893366 1.00412198417076
## 95   0.0008181919340 1.00135522389436 -0.00213657334476
## 65   0.0036221665458 0.99854497560657 -0.00213042814742
## 1    0.9999670233292 -0.00002912561058 0.00005316357863
## 22   0.9999623649516 -0.00002445681020 0.00005315336932
## 111  0.0036736110093 -0.00781186873046 1.00412453478079
## 77  -0.0013175132320 1.00349570753519 -0.00214125396122
## 3    0.9999736641435 -0.00003578128315 0.00005317813266
## 50   0.9999734782302 -0.00003559495390 0.00005317772521
## 28   0.9999675963934 -0.00002969995693 0.00005316483456
## 80   0.0050155487808 0.99714847578679 -0.00212737440765

```

```
## 4 0.9999749482604 -0.00003706827315 0.00005318094693
## 118 -0.0028063209188 -0.00131921054644 1.00411210534781
## 27 0.9999675540962 -0.00002965756509 0.00005316474186
## 51 0.0013112028866 1.00086110986797 -0.00213549286080
## 96 0.0028540313291 0.99931482946630 -0.00213211159437
## 106 -0.0034314274298 -0.00069270540804 1.00411073536291
## 101 -0.0046564853610 0.00053509349531 1.00410805052308
## 84 -0.0089939383462 0.93387668290862 0.07515067271607
## 105 -0.0031998266682 -0.00092482435843 1.00411124293968
## 5 0.9999638725466 -0.00002596777828 0.00005315667337
## 89 0.0023569358584 0.99981303714966 -0.00213320102997
## 129 -0.0023412865148 -0.00178528542896 1.00411312451830
```

```
resultado <- 0
for ( i in 1:dim( pred.nn$net.result )[ 1 ] ) resultado[ i ] <- which.max( pred.nn$net.result[ i, ] )

resultado[ resultado == 1 ] <- "setosa"
resultado[ resultado == 2 ] <- "versicolor"
resultado[ resultado == 3 ] <- "virginica"
resultado
```

```
## [1] "virginica" "setosa" "versicolor" "versicolor" "versicolor"
## [6] "virginica" "versicolor" "versicolor" "versicolor" "versicolor"
## [11] "versicolor" "virginica" "virginica" "setosa" "versicolor"
## [16] "setosa" "versicolor" "versicolor" "setosa" "virginica"
## [21] "setosa" "versicolor" "virginica" "virginica" "versicolor"
## [26] "virginica" "versicolor" "virginica" "versicolor" "versicolor"
## [31] "setosa" "setosa" "virginica" "versicolor" "setosa"
## [36] "setosa" "setosa" "versicolor" "setosa" "virginica"
## [41] "setosa" "versicolor" "versicolor" "virginica" "virginica"
## [46] "versicolor" "virginica" "setosa" "versicolor" "virginica"
```

```
table( Predicción = resultado, Real = iris.test[ , 5 ] )
```

```
##          Real
## Predicción setosa versicolor virginica
## setosa      13         0         0
## versicolor  0         20         2
## virginica   0         0         15
```

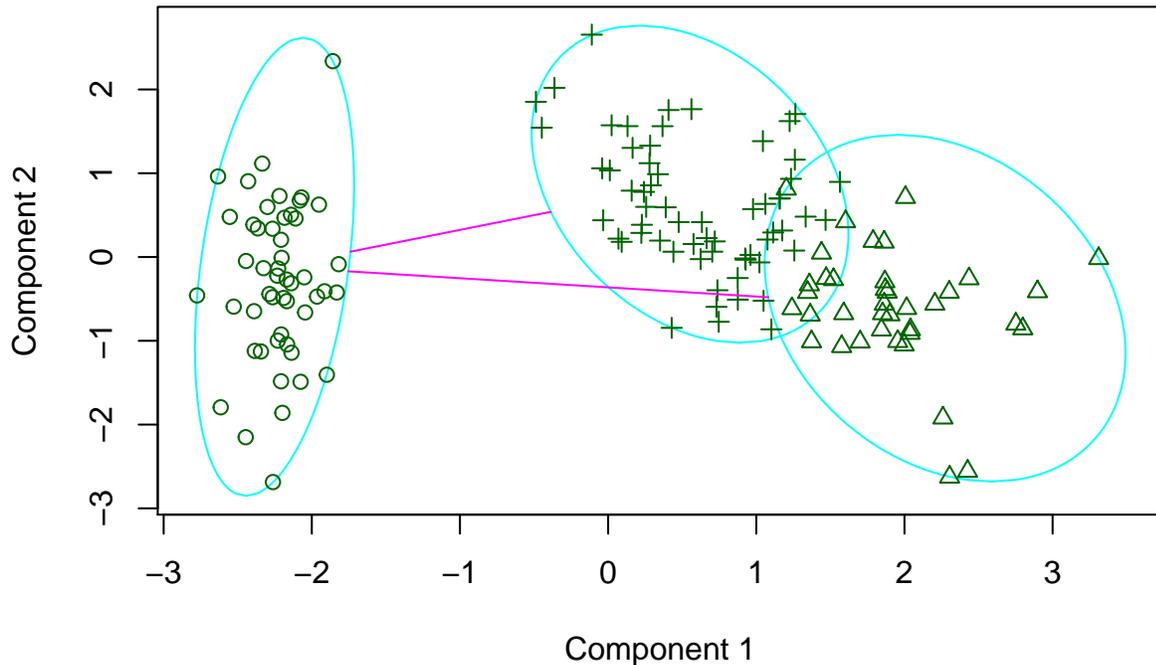
6. Clasificación no supervisada. K-means como clasificador-predictor

En las técnicas multivariantes de clasificación, veremos el clustering no jerárquico o iterativo como técnica para generar grupos de datos en función de sus características.

Ésta es una técnica de clasificación no supervisada pues el clasificador no tiene información sobre las etiquetas de los datos. Genera grupos a los que etiqueta en función de los valores de las variables. Cuando se proporcionan nuevos datos, el clasificador los asignará a los grupos con mayor similitud.

```
mydata <- iris[ , -5 ]
ccl <- cclust( as.matrix( mydata ), 3, 20, method = "kmeans" )
clusplot( mydata, clus = ccl$cluster )
```

CLUSPLOT(mydata)



These two components explain 95.81 % of the point variability.

```
#observaciones extraidas de iris
test <- as.matrix( iris[ sample( c( 1:150 ), 20, replace = FALSE ) , ] )
```

```
#obsevación Simulada
test1 <- matrix( c( 6.6, 3.2, 5.4, 2.4 ), nrow = 1, ncol = 4 )
```

```
# Predicciones
kmPred <- predict( ccl, test[ ,1:4 ] )
kmPred$cluster
```

```
## [1] 3 3 2 1 2 3 2 3 3 3 3 1 3 1 1 1 3 2 2 3
```

```
kmPred
```

```
##
##                               Clustering on Test Set
##
##
## Number of Clusters: 3
## Sizes of Clusters: 5 5 10
```

Comprobar los resultados

```
kmPredLabel <- factor( kmPred$cluster, labels = c( "setosa", "versicolor", "virginica" ) )
table( Predicción = kmPredLabel, Real = test[ , 5 ] )
```

```
##           Real
## Predicción setosa versicolor virginica
## setosa      5         0         0
## versicolor  0         0         5
## virginica   0         8         2
```

7. Evaluar la eficacia del modelo

Vamos a estudiar algunos de los mecanismos existentes de validación de la clasificación realizada por nuestro modelo predictivo. para ello vamos a generar un modelo nuevo con datos de pacientes que tienen un tumor, cuyo diagnóstico está etiquetado como benigno o maligno.

La primera columna contiene un identificador del sujeto, la segunda corresponde al diagnóstico y el resto son variables medidas al tumor.

```
wdbc <- read.csv( "http://ares.inf.um.es/00RTeam/datos/wdbc.csv" )
```

```
head( wdbc )
```

```
##      id diagnosis radius_mean texture_mean perimeter_mean area_mean
## 1  842302         M      17.99      10.38         122.80      1001.0
## 2  842517         M      20.57      17.77         132.90      1326.0
## 3 84300903         M      19.69      21.25         130.00      1203.0
## 4 84348301         M      11.42      20.38          77.58       386.1
## 5 84358402         M      20.29      14.34         135.10      1297.0
## 6  843786         M      12.45      15.70          82.57       477.1
## smoothness_mean compactness_mean concavity_mean concave.points_mean
## 1      0.11840      0.27760      0.3001      0.14710
## 2      0.08474      0.07864      0.0869      0.07017
## 3      0.10960      0.15990      0.1974      0.12790
## 4      0.14250      0.28390      0.2414      0.10520
## 5      0.10030      0.13280      0.1980      0.10430
## 6      0.12780      0.17000      0.1578      0.08089
## symmetry_mean fractal_dimension_mean radius_se texture_se perimeter_se
## 1      0.2419      0.07871      1.0950      0.9053      8.589
## 2      0.1812      0.05667      0.5435      0.7339      3.398
## 3      0.2069      0.05999      0.7456      0.7869      4.585
## 4      0.2597      0.09744      0.4956      1.1560      3.445
## 5      0.1809      0.05883      0.7572      0.7813      5.438
## 6      0.2087      0.07613      0.3345      0.8902      2.217
## area_se smoothness_se compactness_se concavity_se concave.points_se
## 1 153.40      0.006399      0.04904      0.05373      0.01587
## 2  74.08      0.005225      0.01308      0.01860      0.01340
## 3  94.03      0.006150      0.04006      0.03832      0.02058
## 4  27.23      0.009110      0.07458      0.05661      0.01867
## 5  94.44      0.011490      0.02461      0.05688      0.01885
## 6  27.19      0.007510      0.03345      0.03672      0.01137
## symmetry_se fractal_dimension_se radius_worst texture_worst
## 1  0.03003      0.006193      25.38      17.33
## 2  0.01389      0.003532      24.99      23.41
## 3  0.02250      0.004571      23.57      25.53
## 4  0.05963      0.009208      14.91      26.50
## 5  0.01756      0.005115      22.54      16.67
## 6  0.02165      0.005082      15.47      23.75
## perimeter_worst area_worst smoothness_worst compactness_worst
## 1 184.60      2019.0      0.1622      0.6656
## 2 158.80      1956.0      0.1238      0.1866
## 3 152.50      1709.0      0.1444      0.4245
## 4  98.87      567.7      0.2098      0.8663
## 5 152.20      1575.0      0.1374      0.2050
## 6 103.40      741.6      0.1791      0.5249
```

```
## concavity_worst concave.points_worst symmetry_worst
## 1 0.7119 0.2654 0.4601
## 2 0.2416 0.1860 0.2750
## 3 0.4504 0.2430 0.3613
## 4 0.6869 0.2575 0.6638
## 5 0.4000 0.1625 0.2364
## 6 0.5355 0.1741 0.3985
## fractal_dimension_worst
## 1 0.11890
## 2 0.08902
## 3 0.08758
## 4 0.17300
## 5 0.07678
## 6 0.12440
```

```
#eliminar columna de ID
```

```
wdbc <- wdbc[ -1 ]
```

```
# Normalizar los datos
```

```
wdbcN <- scale( wdbc[ 2:31 ], scale = TRUE, center = TRUE )
```

7.0.1. Crear datos de entrenamiento y datos de test

```
# Datos
```

```
wTrain <- wdbcN[ 1:469, ] # Los 469 primeros registros
```

```
wTest <- wdbcN[ 470:569, ] # Los últimos 100 registros
```

```
# Etiquetas
```

```
wTrainLabel <- wdbc[ 1:469, 1 ]
```

```
wTestLabel <- wdbc[ 470:569, 1 ]
```

7.0.2. Entrenar el modelo

```
wPredict<- knn( wTrain, wTest, cl = wTrainLabel, k = 21, prob = TRUE )
```

```
wPredict
```

```
## [1] B B B B B B B B B M B B B B B B M B B B B M B B B B M M B M B M
## [36] B B B B B M B B M B B B M M B B B M B B B B B B B B B M B M M B B
## [71] B B B B B B B B B B B B B B B B B B B B B B M M M M M M B
## attr(,"prob")
## [1] 0.7619047619 1.0000000000 0.8571428571 1.0000000000 0.9523809524
## [6] 0.9523809524 1.0000000000 0.8571428571 0.9523809524 1.0000000000
## [11] 0.7619047619 1.0000000000 0.9047619048 0.9523809524 1.0000000000
## [16] 0.7142857143 0.8571428571 0.9047619048 1.0000000000 1.0000000000
## [21] 0.6190476190 0.9047619048 0.8571428571 1.0000000000 1.0000000000
## [26] 1.0000000000 0.8095238095 0.6190476190 1.0000000000 1.0000000000
## [31] 1.0000000000 0.9523809524 0.9523809524 0.9523809524 1.0000000000
## [36] 0.6666666667 0.7142857143 1.0000000000 1.0000000000 0.9523809524
## [41] 1.0000000000 1.0000000000 1.0000000000 0.9523809524 1.0000000000
## [46] 0.6666666667 1.0000000000 1.0000000000 1.0000000000 0.8571428571
## [51] 0.9523809524 1.0000000000 1.0000000000 1.0000000000 0.8571428571
## [56] 1.0000000000 1.0000000000 0.6190476190 1.0000000000 0.8571428571
```

```
## [61] 1.0000000000 0.9523809524 0.9047619048 1.0000000000 1.0000000000
## [66] 1.0000000000 1.0000000000 0.5238095238 0.6666666667 1.0000000000
## [71] 0.9523809524 1.0000000000 0.5714285714 0.6190476190 0.8571428571
## [76] 0.9523809524 0.8571428571 1.0000000000 1.0000000000 1.0000000000
## [81] 1.0000000000 1.0000000000 1.0000000000 0.8095238095 1.0000000000
## [86] 0.8571428571 0.9047619048 1.0000000000 1.0000000000 0.9523809524
## [91] 0.8095238095 0.7619047619 1.0000000000 1.0000000000 1.0000000000
## [96] 1.0000000000 1.0000000000 0.7619047619 1.0000000000 1.0000000000
## Levels: B M
```

```
# length(wKnn)
```

7.0.3. Evaluar el modelo

7.0.3.1. Simple tabla con table()

```
table( Predicción = wPredict, Observación = wTestLabel )
```

```
##           Observación
## Predicción B M
##           B 77 2
##           M 0 21
```

7.0.3.2. Con la función CrossTable()

Calcula una tabla similar a la anterior pero más completa. Aporta proporciones de aciertos por filas, columnas y totales.

```
CrossTable( x = wPredict, y = wTestLabel, dnn = c( "Predicción", "Observación" ),
            prop.chisq = FALSE )
```

```
##
##
## Cell Contents
## |-----|
## |                N |
## |      N / Row Total |
## |      N / Col Total |
## |      N / Table Total |
## |-----|
##
##
## Total Observations in Table: 100
##
##
##           | Observación
## Predicción |          B |          M | Row Total |
## -----|-----|-----|-----|
##           B |          77 |           2 |          79 |
##           |          0.975 |          0.025 |          0.790 |
##           |          1.000 |          0.087 |          |
##           |          0.770 |          0.020 |          |
## -----|-----|-----|-----|
##           M |           0 |          21 |          21 |
##           |          0.000 |          1.000 |          0.210 |
```

```
##          |      0.000 |      0.913 |          |
##          |      0.000 |      0.210 |          |
## -----|-----|-----|-----|
## Column Total |      77 |      23 |      100 |
##          |      0.770 |      0.230 |          |
## -----|-----|-----|-----|
##
##
```

7.0.3.3. Validación cruzada (cross-validation)

La función `knn.cv()` entrena al clasificador, hace el test los mismos datos y realiza una validación cruzada al vuelo

```
knn.cv <- knn.cv( train = wTest, cl = wTestLabel, k = 21, prob = TRUE )
knn.cv
```

```
## [1] B B B B B B B B B M B B B B B B M B B B B M B B B B M M B B B M
## [36] B B B B B M B B B B B M M B B B M B B B B B B B B B M B M B B B
## [71] B B B B B B B B B B B B B B B B B B B M M M M B M B
## attr(,"prob")
## [1] 0.8095238095 1.0000000000 1.0000000000 0.9523809524 1.0000000000
## [6] 1.0000000000 1.0000000000 0.9047619048 0.9523809524 1.0000000000
## [11] 0.6190476190 0.9523809524 0.9523809524 1.0000000000 0.9523809524
## [16] 0.9523809524 0.6666666667 0.9523809524 0.8571428571 1.0000000000
## [21] 0.9523809524 0.9523809524 0.9523809524 0.6190476190 1.0000000000
## [26] 0.9523809524 0.9047619048 0.9523809524 0.9523809524 0.6666666667
## [31] 0.8571428571 0.9523809524 0.6666666667 1.0000000000 0.7619047619
## [36] 0.8571428571 0.8095238095 1.0000000000 1.0000000000 0.9523809524
## [41] 0.5714285714 1.0000000000 0.9523809524 0.6666666667 0.9523809524
## [46] 0.9523809524 1.0000000000 0.5714285714 0.6190476190 1.0000000000
## [51] 1.0000000000 0.9523809524 0.9047619048 1.0000000000 0.9523809524
## [56] 1.0000000000 1.0000000000 1.0000000000 1.0000000000 0.9047619048
## [61] 1.0000000000 0.9523809524 1.0000000000 0.9523809524 0.7619047619
## [66] 1.0000000000 0.8095238095 0.8571428571 0.8095238095 1.0000000000
## [71] 0.9523809524 1.0000000000 0.7142857143 0.8571428571 0.9523809524
## [76] 0.9523809524 0.9523809524 1.0000000000 1.0000000000 1.0000000000
## [81] 1.0000000000 1.0000000000 1.0000000000 0.9523809524 1.0000000000
## [86] 0.8571428571 0.9523809524 1.0000000000 1.0000000000 0.8571428571
## [91] 0.9047619048 0.8571428571 1.0000000000 0.7142857143 0.9047619048
## [96] 0.8095238095 0.6190476190 0.7142857143 0.9047619048 1.0000000000
## Levels: B M
```

```
table( Predicción = knncv, Observación = wTestLabel)
```

```
##          Observación
## Predicción B M
##          B 77 6
##          M 0 17
```

Con el paquete `caret`

```
confusionMatrix( knncv, wTestLabel, positive = "M", dnn = "" )
```

```
## Confusion Matrix and Statistics
##
## NA
```

```

##      B  M
##    B 77  6
##    M  0 17
##
##          Accuracy : 0.94
##          95% CI : (0.8739701, 0.9776651)
##    No Information Rate : 0.77
##    P-Value [Acc > NIR] : 0.000004732313
##
##          Kappa : 0.8135488
## Mcnemar's Test P-Value : 0.04122683
##
##          Sensitivity : 0.7391304
##          Specificity : 1.0000000
##    Pos Pred Value : 1.0000000
##    Neg Pred Value : 0.9277108
##          Prevalence : 0.2300000
##    Detection Rate : 0.1700000
##    Detection Prevalence : 0.1700000
##    Balanced Accuracy : 0.8695652
##
##    'Positive' Class : M
##
##

```

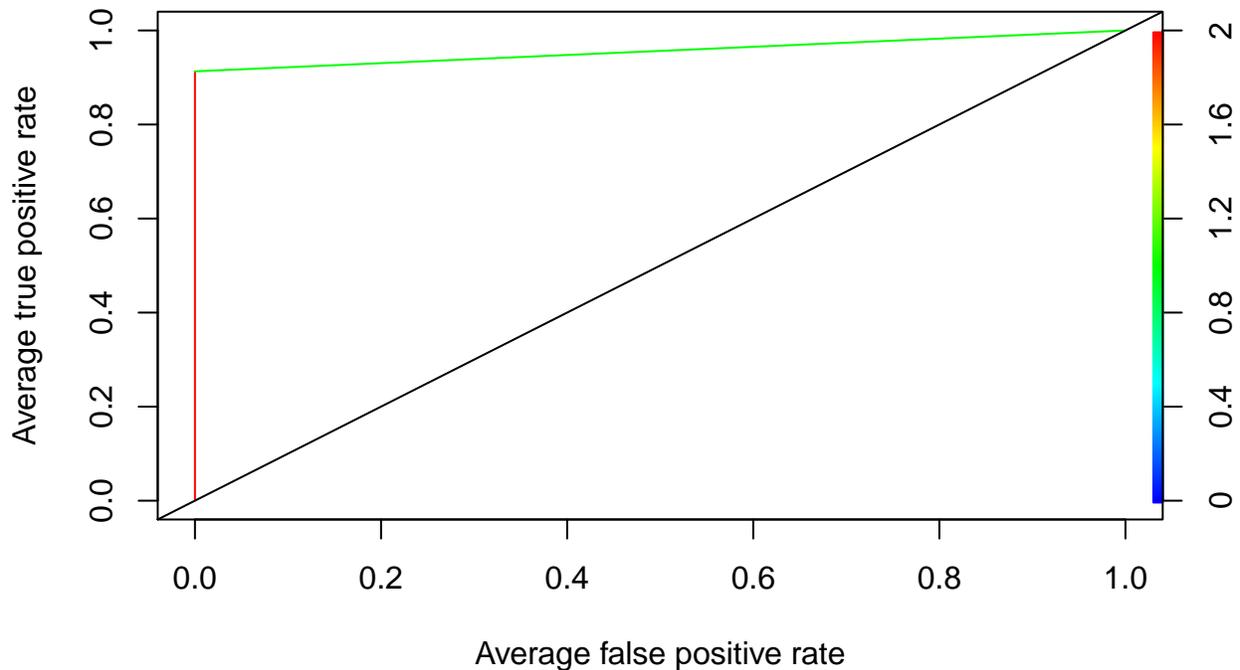
7.0.3.4. Curvas ROC (*Receiver Operating Characteristic*)

Es una representación gráfica de la *sensibilidad* frente a $1 - \textit{especificidad}$ para un clasificador binario, es decir sólo hay dos respuestas, positivo y negativo.

```

prd <- ifelse( wPredict == "M", 1, 0 )
real <- ifelse( wTestLabel == "M", 1, 0 )
predKnn <- ROCR::prediction( prd, real )
perf <- performance( predKnn, measure = "tpr", x.measure = "fpr" )
plot( perf, avg = "threshold", colorize = T )
abline( a = 0, b = 1 )

```



```
# área bajo la curva
AUC <- performance( predKnn, measure = "auc" )
AUCtumor <- AUC@y.values
paste( "Área bajo la curva: ", round( AUCtumor[[ 1 ]], 4 ) )

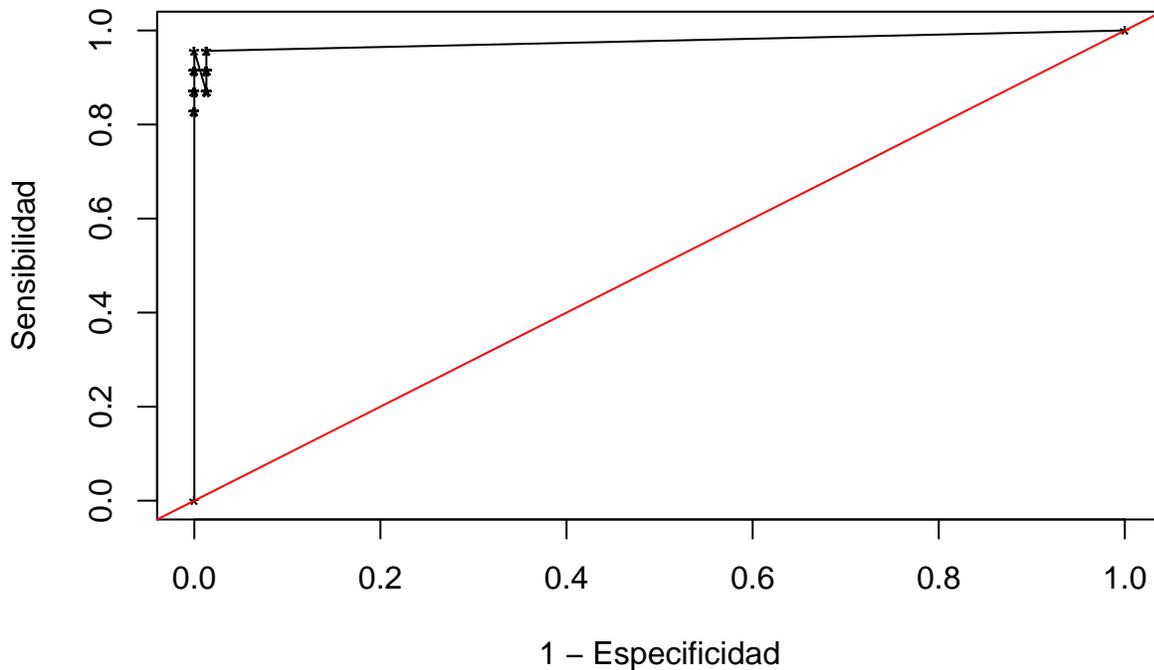
## [1] "Área bajo la curva: 0.9565"

Variar parámetro k (número de vecinos)

knnPerf <- function (cost = 1){
  wPredict <- knn( wTrain, wTest, cl = wTrainLabel, k = cost, prob = TRUE )
  prd <- ifelse( wPredict == "M", 1, 0 )
  real <- ifelse( wTestLabel == "M", 1, 0 )
  predKnn <- ROCR::prediction( prd, real )
  perf <- performance( predKnn, measure = "tpr", x.measure = "fpr" )
  data.frame( fpr = perf@x.values[[ 1 ]][ 2 ],
             tpr = perf@y.values[[ 1 ]][ 2 ],
             threshold = perf@alpha.values[[ 1 ]],
             k = cost ) }

df <- data.frame ()
for (cost in 10:200){
  df <- rbind ( df, knnPerf( cost ) ) }

df2 <- df[with( df, order( fpr, tpr ) ),]
df3 <- rbind( c( 0, 0, 0, 0 ), df2, c( 1, 1, 0, 0 ) )
plot ( df3$fpr, df3$tpr, type = "l", xlab = "1 - Especificidad", ylab = "Sensibilidad" )
points( df3$fpr, df3$tpr, pch = "*" )
abline( a = 0, b = 1, col = "red" )
```



Alternativa al cálculo de parámetros de la curva ROC

```
dfAlt <- data.frame()
for ( cost in 10:300 ){
  wPredict<- knn( wTrain, wTest, cl = wTrainLabel, k = cost, prob = TRUE )
  aux <- t( as.data.frame( ( confusionMatrix( wPredict, wTestLabel,
                                             positive = "M", dnn = "" ) )$byClass[ 1:2 ] ) )
  rownames( aux ) <- NULL
  dfAlt <- rbind( dfAlt, aux )
}

dfAlt$Specificity <- 1 - dfAlt$Specificity
dfAlt2 <- dfAlt[ with( dfAlt, order( Specificity, Sensitivity ) ), ]
dfAlt3 <- rbind( c( 0, 0), dfAlt2, c( 1, 1) )
plot ( dfAlt3$Specificity, dfAlt3$Sensitivity, type = "l",
       xlab = "1 - Especificidad", ylab = "Sensibilidad" )
points( dfAlt3$Specificity, dfAlt3$Sensitivity, pch = "*" )
abline( a = 0, b = 1 , col = "red" )
```

