Machine Learning

00R Team

Abril 2016

Índice

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....

```
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
```

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]
```

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

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

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

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

Validar resultados

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

```
##
## iris.pred    setosa versicolor virginica
##   setosa         13          0         0
##   versicolor      0          8         1
##   virginica       0          0         8
```

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

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

mi_predict
```

```
## [1] setosa
## 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 que mayor probabilidad de pertenecia.

Ejemplo en R

```
data(iris)
```

Preprocesar datos:

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

```
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
```

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, kernel = "radial")
summary(svm_model)
```

```
##
## Call:
## svm(formula = Species ~ ., data = iris.train, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.25
##
## Number of Support Vectors:  43
##
##  ( 18 7 18 )
##
##
## 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])
SVM_predict
```

```
##        103         64         83        120        100         61
##  virginica versicolor versicolor  virginica versicolor versicolor
##        129         29        109         15        134         17
##  virginica     setosa  virginica     setosa versicolor     setosa
##         30         55        117         38         49         22
```

```
##     setosa versicolor  virginica     setosa     setosa     setosa
##         32        126        106          6         77        148
##     setosa  virginica  virginica     setosa versicolor  virginica
##         41         14        113          1        135        140
##     setosa     setosa  virginica     setosa  virginica  virginica
## Levels: setosa versicolor virginica
```

Contrastar la predicción con la realidad

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

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

```
# Versión más completa
CrossTable(x = iris.test[,5], y = SVM_predict, prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  30
##
##
##                | SVM_predict
## iris.test[, 5] |     setosa | versicolor |  virginica |  Row Total |
## ---------------|------------|------------|------------|------------|
##         setosa |         12 |          0 |          0 |         12 |
##                |      1.000 |      0.000 |      0.000 |      0.400 |
##                |      1.000 |      0.000 |      0.000 |            |
##                |      0.400 |      0.000 |      0.000 |            |
## ---------------|------------|------------|------------|------------|
##     versicolor |          0 |          6 |          0 |          6 |
##                |      0.000 |      1.000 |      0.000 |      0.200 |
##                |      0.000 |      0.857 |      0.000 |            |
##                |      0.000 |      0.200 |      0.000 |            |
## ---------------|------------|------------|------------|------------|
##      virginica |          0 |          1 |         11 |         12 |
##                |      0.000 |      0.083 |      0.917 |      0.400 |
##                |      0.000 |      0.143 |      1.000 |            |
##                |      0.000 |      0.033 |      0.367 |            |
## ---------------|------------|------------|------------|------------|
```

```
##   Column Total |         12 |          7 |         11 |         30 |
##                |      0.400 |      0.233 |      0.367 |            |
## ---------------|------------|------------|------------|------------|
##
##
```

Y con mi muestra encontrada en el campo

```
mi_iris<-matrix(c(5.0, 3.5, 1.3, 0.1), nrow=1, ncol=4)
colnames(mi_iris)<-colnames(iris[,1:4])
# predict requiere un matrix o data.frame no un vector
predict(svm_model, mi_iris)
```

```
##      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
##       0.29       0.33       0.38
##
## Conditional probabilities:
##             Sepal.Length
## Y                [,1]      [,2]
##   setosa     5.003448 0.3417212
##   versicolor 5.990909 0.5558572
##   virginica  6.515789 0.6482935
##
##             Sepal.Width
## Y                [,1]      [,2]
##   setosa     3.379310 0.3519124
##   versicolor 2.833333 0.2813657
##   virginica  2.928947 0.2994661
```

```
##
##             Petal.Length
## Y                [,1]      [,2]
##   setosa     1.441379 0.1701202
##   versicolor 4.293939 0.4980834
##   virginica  5.500000 0.5550749
##
##             Petal.Width
## Y                [,1]      [,2]
##   setosa     0.262069 0.1177582
##   versicolor 1.336364 0.1884446
##   virginica  2.005263 0.2799106
```

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

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

Contrastar la predicción

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

```
##
## predNB       setosa versicolor virginica
##   setosa         12          0         0
##   versicolor      0          6         2
##   virginica       0          0        10
```

Y con mi muestra encontrada en el campo

```
mi_iris<-matrix(c(5.0, 3.5, 1.3, 0.1), nrow=1, ncol=4)
colnames(mi_iris)<-colnames(iris[,1:4])
# predict requiere un matrix o data.frame no un vector
predict(iris.NB, mi_iris)
```

```
## [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, esta 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....

```
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
```

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]
```

Binarizar las categorias

```
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
## 34  1          5.5         4.2          1.4         0.2
## 74  1          6.1         2.8          4.7         1.2
## 111 1          6.5         3.2          5.1         2.0
## 56  1          5.7         2.8          4.5         1.3
## 108 1          7.3         2.9          6.3         1.8
## 112 1          6.4         2.7          5.3         1.9
## 117 1          6.5         3.0          5.5         1.8
## 68  1          5.8         2.7          4.1         1.0
## 20  1          5.1         3.8          1.5         0.3
## 14  1          4.3         3.0          1.1         0.1
## 58  1          4.9         2.4          3.3         1.0
## 102 1          5.8         2.7          5.1         1.9
## 44  1          5.0         3.5          1.6         0.6
## 15  1          5.8         4.0          1.2         0.2
## 140 1          6.9         3.1          5.4         2.1
## 129 1          6.4         2.8          5.6         2.1
## 30  1          4.7         3.2          1.6         0.2
## 126 1          7.2         3.2          6.0         1.8
## 49  1          5.3         3.7          1.5         0.2
## 66  1          6.7         3.1          4.4         1.4
## 133 1          6.4         2.8          5.6         2.2
## 65  1          5.6         2.9          3.6         1.3
## 21  1          5.4         3.4          1.7         0.2
## 63  1          6.0         2.2          4.0         1.0
## 69  1          6.2         2.2          4.5         1.5
## 94  1          5.0         2.3          3.3         1.0
## 95  1          5.6         2.7          4.2         1.3
## 85  1          5.4         3.0          4.5         1.5
## 4   1          4.6         3.1          1.5         0.2
## 17  1          5.4         3.9          1.3         0.4
##
## $neurons[[2]]
##     [,1]                [,2]         [,3]           [,4]
## 34     1 0.00000000623572993 0.6388400994 0.005400215431
## 74     1 0.05434265403658232 0.3290765504 0.386940106096
## 111    1 0.99956146944692981 0.3114174859 0.613976588222
## 56     1 0.27961859202047468 0.3147445134 0.387290505777
## 108    1 0.98272283482190459 0.2361226007 0.734259355738
## 112    1 0.99943475754159894 0.3207642178 0.763701935164
## 117    1 0.99238249605968643 0.2533756250 0.613391684133
## 68     1 0.00698180149481541 0.4039355123 0.314804804937
## 20     1 0.00000009945515659 0.6215321827 0.011410713944
## 14     1 0.00000009368556929 0.6788833354 0.024286670517
## 58     1 0.04965696523029048 0.4728034528 0.349271607865
## 102    1 0.99964839435390818 0.2720968601 0.709426408455
## 44     1 0.00002218136825888 0.6515621687 0.032916797383
## 15     1 0.00000001130442683 0.7462014906 0.009084657340
```

```
## 140    1 0.99988628038147920 0.3342930524 0.735970874876
## 129    1 0.99995437705942913 0.2660794967 0.794722517844
## 30     1 0.00000012969203573 0.6192094867 0.024043407861
## 126    1 0.97105539144916220 0.2368675157 0.600066779068
## 49     1 0.00000002576664865 0.6603333849 0.012553063031
## 66     1 0.34029754523528039 0.4582721674 0.409746677389
## 133    1 0.99999015917101330 0.2722890033 0.823612894386
## 65     1 0.40772022268317665 0.4702961830 0.336071445213
## 21     1 0.00000004777629493 0.6788638600 0.022911355367
## 63     1 0.02773482933836232 0.5384706760 0.565954829267
## 69     1 0.97251942817136339 0.5022571217 0.783945202472
## 94     1 0.06185685132359567 0.5057840765 0.404783998865
## 95     1 0.41421175130649429 0.3731296334 0.422682780732
## 85     1 0.84626126518001876 0.2597593948 0.353299645498
## 4      1 0.00000019753628655 0.6386730688 0.027656678430
## 17     1 0.00000035878579167 0.6993964539 0.013143003067
##
## $neurons[[3]]
##     [,1]
## 34     1
## 74     1
## 111    1
## 56     1
## 108    1
## 112    1
## 117    1
## 68     1
## 20     1
## 14     1
## 58     1
## 102    1
## 44     1
## 15     1
## 140    1
## 129    1
## 30     1
## 126    1
## 49     1
## 66     1
## 133    1
## 65     1
## 21     1
## 63     1
## 69     1
## 94     1
## 95     1
## 85     1
## 4      1
## 17     1
##                                                                          [,2]
## 34  0.000000000000000000000000000000000000000000000000000001196058027198950443
## 74  0.00000561560144012085741497403157396206552220974117517471313476562500000000
## 111 0.99996195807930421395326447964180260896682739257812500000000000000000000
## 56  0.00000576722719173994477720544132859359365284035447984933853149414062500000
```

```
## 108 1.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
## 112 0.999999999999621413948602821619715541601181030273437500000000000000000000000000000000000000
## 117 0.999999999596671220410689784330315887928009033203125000000000000000000000000000000000000000
## 68  0.000000000000000226458620203221601163009360115744124259540151836156418597
## 20  0.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
## 14  0.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
## 58  0.000000000000000000017643676437770122579772577815736058950053900680160622
## 102 0.999999999999960254015718419395852833986282348632812500000000000000000000000000000000000000
## 44  0.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
## 15  0.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
## 140 0.999999999765592839651162648806348443031311035156250000000000000000000000000000000000000000
## 129 1.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
## 30  0.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
## 126 0.999999999928496752232831568107940256595611572265625000000000000000000000000000000000000000
## 49  0.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
## 66  0.000000000000000028250674949012892227171178923044723028210863178654593808
## 133 1.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
## 65  0.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
## 21  0.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
## 63  0.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
## 69  0.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
## 94  0.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
## 95  0.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
## 85  0.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
## 4   0.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
## 17  0.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
##                 [,3]
## 34   5.381418123e-01
## 74   2.202943658e-10
## 111 1.106601663e-179
## 56   9.824235082e-51
## 108 1.156838644e-176
## 112 1.176034486e-179
## 117 2.137918448e-178
## 68   6.253244936e-02
## 20   5.382421343e-01
## 14   5.383492572e-01
## 58   1.517059194e-09
## 102 1.073764415e-179
## 44   5.362456961e-01
## 15   5.380476714e-01
## 140 9.745787940e-180
## 129 9.511646147e-180
## 30   5.384234966e-01
## 126 1.411702193e-174
## 49   5.382131379e-01
## 66   1.330112307e-61
## 133 9.387635105e-180
## 65   1.110363283e-73
## 21   5.383342524e-01
## 63   1.296884223e-05
## 69  7.789107857e-175
## 94   9.933656939e-12
## 95   7.675836661e-75
```

```
## 85  3.135077185e-152
## 4    5.384418388e-01
## 17   5.381341653e-01
##
##
## $net.result
##                    [,1]              [,2]               [,3]
## 34   0.9998455557388730  0.00015945007747  0.000001880163389
## 74  -0.0000762043977547  1.00254185906816 -0.002462411828445
## 111  0.0000002374837031  0.02684159726079  0.973164745399927
## 56  -0.0000762047954927  1.00254171153075 -0.002462263892796
## 108  0.0000002403918417  0.02680447812826  0.973201861751279
## 112  0.0000002403918416  0.02680447812863  0.973201861750910
## 117  0.0000002403918111  0.02680447851889  0.973201861360687
## 68   0.1161153923159505  0.88606917555869 -0.002180901714129
## 20   1.0000319641674966 -0.00002741809414  0.000002340585537
## 14   1.0002310091956388 -0.00022695403117  0.000002832219633
## 58  -0.0000762024175230  1.00254733603571 -0.002467890794961
## 102  0.0000002403918417  0.02680447812830  0.973201861751240
## 44   0.9963223806195726  0.00369131447674 -0.000006821953071
## 15   0.9996706324417652  0.00033480479153  0.000001448109105
## 140  0.0000002403918237  0.02680447835698  0.973201861522575
## 129  0.0000002403918417  0.02680447812826  0.973201861751279
## 30   1.0003689534746252 -0.00036523852499  0.000003172937066
## 126  0.0000002403918362  0.02680447819803  0.973201861681516
## 49   0.9999780859172410  0.00002659303717  0.000002207508187
## 66  -0.0000762052363720  1.00254733886151 -0.002467890801923
## 133  0.0000002403918417  0.02680447812826  0.973201861751279
## 65  -0.0000762052363720  1.00254733886151 -0.002467890801923
## 21   1.0002031287613791 -0.00019900483481  0.000002763355958
## 63  -0.0000521078195509  1.00252318201243 -0.002467831281872
## 69  -0.0000749849286688  0.98697147720540  0.013106803822432
## 94  -0.0000762052179143  1.00254733884301 -0.002467890801878
## 95  -0.0000762052362757  1.00254733763197 -0.002467889572482
## 85  -0.0000762050603292  1.00254509187183 -0.002465643980604
## 4    1.0004030352044004 -0.00039940431125  0.000003257117719
## 17   0.9998313467935185  0.00017369406664  0.000001845067803
```

```
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] "setosa"     "versicolor" "virginica"  "versicolor" "virginica"
##  [6] "virginica"  "virginica"  "versicolor" "setosa"     "setosa"
## [11] "versicolor" "virginica"  "setosa"     "setosa"     "virginica"
## [16] "virginica"  "setosa"     "virginica"  "setosa"     "versicolor"
## [21] "virginica"  "versicolor" "setosa"     "versicolor" "versicolor"
## [26] "versicolor" "versicolor" "versicolor" "setosa"     "setosa"
```

```r
table(resultado, iris.test[,5])
```

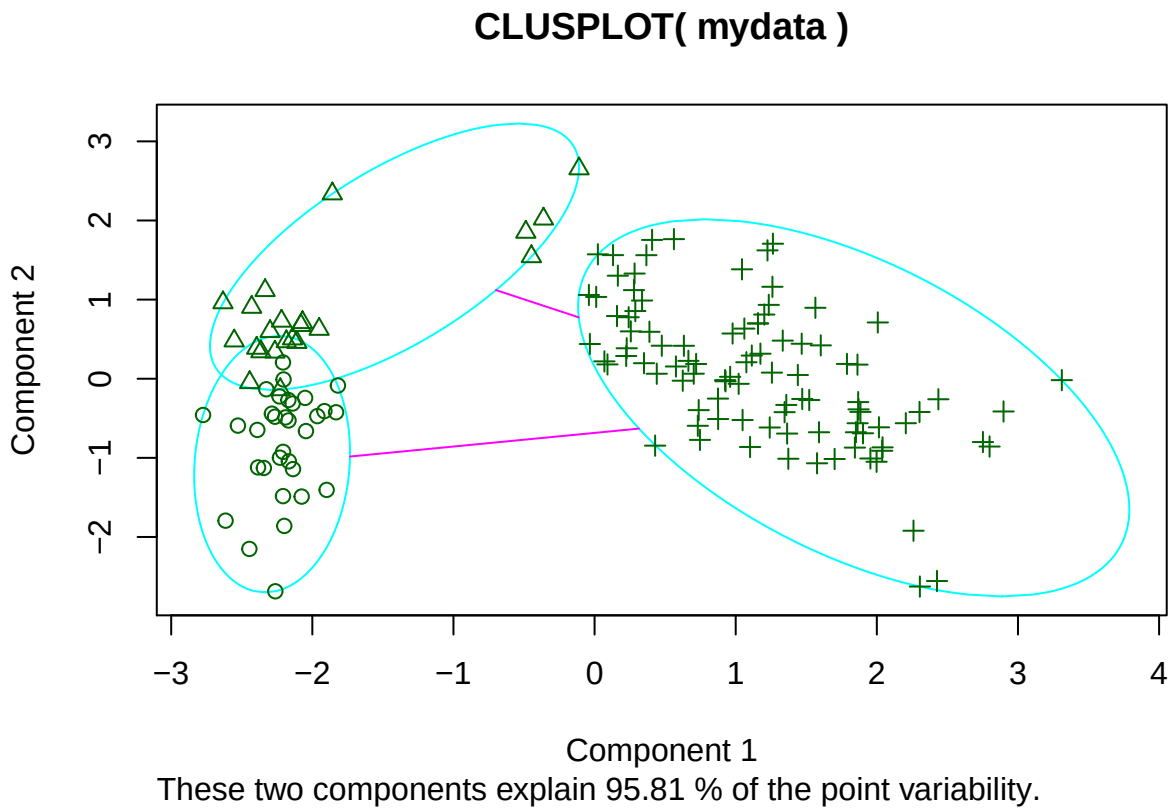
```
##
## resultado    setosa versicolor virginica
##   setosa         10          0         0
##   versicolor      0         11         0
##   virginica       0          0         9
```

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

En las técnicas multivariantes de clasificación, vimos 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.

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




```r
#observaciones extraidas de iris
test<- as.matrix(iris[c(1:5,55:60,135:140),1:5])
#obsevación inventada
```

```r
test1<-matrix(c(6.6,3.2,5.4,2.4), nrow=1, ncol=4)
# Prediciones
kmPred<-predict(ccl, test[,1:4])
kmPred$cluster
```

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

```r
kmPred
```

```
##
##                               Clustering on Test Set
##
##
## Number of Clusters:  3
## Sizes  of Clusters:  2 4 11
```

Comprobar los resultados

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

```
##
## kmPredLabel  setosa versicolor virginica
##   setosa          2          0         0
##   versicolor      3          1         0
##   virginica       0          5         6
```

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.

```r
wdbc <- read.csv("~/007cursodocint/1516/man01/MatyMet/taller3/wdbc.csv")
#
# head(wdbc)
#
#eliminar columna de ID.
wdbc<-wdbc[-1]
# Normalizar los datos
normalize<-{function(x)
  return ( (x-min(x) ) / (max(x)-min(x) ) )
  }
wdbc_N<- as.data.frame(lapply(wdbc[2:31], normalize))
```

7.0.2. Crear datos de entrenamiento y datos de test

```r
# Datos
w_train<- wdbc_N[1:469, ] # Los 469 primeros registros
w_test<- wdbc_N[470:569, ] # Los últimos 100 registros
# Etiquetas
w_train_label <- wdbc[1:469, 1]
w_test_label <- wdbc[470:569, 1]
```

7.0.3. Entrenar el modelo

```r
w_predict<- knn(w_train, w_test, cl = w_train_label, k = 21, prob = TRUE )
w_predict
```

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

```r
# length(w_knn)
```

7.0.4. Evaluar el modelo

7.0.4.1. Simple tabla con `table()`

```r
table(w_predict, w_test_label)
```

```
##          w_test_label
## w_predict  B  M
##         B 77  2
##         M  0 21
```

7.0.4.2. 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 = w_test_label, y = w_predict, prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  100
##
##
##              | w_predict
## w_test_label |         B |         M | Row Total |
## -------------|-----------|-----------|-----------|
##            B |        77 |         0 |        77 |
##              |     1.000 |     0.000 |     0.770 |
##              |     0.975 |     0.000 |           |
##              |     0.770 |     0.000 |           |
## -------------|-----------|-----------|-----------|
##            M |         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 |           |
## -------------|-----------|-----------|-----------|
##
##
```

7.0.4.3. Validación cruzada (cross-validation)

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

```
knncv<-knn.cv(train = w_test, cl = w_test_label, k=10, prob = TRUE)
knncv
```

```
##   [1] B B B B B B B B B M B B B B B B B M B B B B M B B B B B M M B B B M
##  [36] B B B B B M B B B B B B M M B B B M B B 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 B B B M M M M B M B
## attr(,"prob")
##   [1] 0.9 1.0 1.0 1.0 1.0 1.0 1.0 0.9 1.0 1.0 0.8 1.0 0.9 1.0 0.9 1.0 0.7
```

```
##  [18] 0.9 1.0 1.0 0.9 1.0 0.9 1.0 1.0 1.0 0.9 1.0 1.0 0.8 1.0 0.9 0.6 1.0
##  [35] 1.0 0.9 0.9 1.0 1.0 0.9 0.7 1.0 1.0 0.6 0.9 1.0 1.0 1.0 1.0 1.0 1.0
##  [52] 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 0.8
##  [69] 0.9 1.0 1.0 1.0 0.7 0.9 1.0 0.9 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
##  [86] 1.0 1.0 1.0 1.0 0.7 0.9 1.0 1.0 0.9 1.0 1.0 1.0 0.8 1.0 1.0
## Levels: B M
```

```
tb<-table( w_test_label, knncv)
```

Tenemos 4 situaciones:

- TP: verdaderos positivos. Observaciones clasificadas como tumor maligno y realmente lo es.
- TN: verdaderos negativos. Observaciones clasificadas como tumor benigno y el benigno.
- FP: Falsos positivos. Observaciones clasificadas como malignos, siendo benignos.
- FN: Falsos negativos. Observaciones clasificadas como benignos, siendo malignos.

La precisión (o ratio de aciertos) de nuestra matriz de confusión se calcula como:

$$Precisión = \frac{TP + TN}{TP + TN + FP + FN}$$

El ratio de error o proporción de clasificados incorrectamente:

$$ratiodeerror = 1 - precisión$$

En nuestro caso la precisión sería:

```
TP<-tb[2,2]
TN<-tb[1,1]
FN<-tb[2,1]
FP<-tb[1,2]
accur<- (TP+TN)/(TP+TN+FP+FN)
accur
```

```
## [1] 0.94
```

```
error_rate <- 1 - accur
error_rate
```

```
## [1] 0.06
```

Otros términos:

- Sensibilidad. Ratio de verdaderos positivos, es el cociente entre los verdaderos positivos y el total de positivos. La mayoría de tumores malignos (positivos) se clasifiquen como tales.

$$sensibilidad = \frac{TP}{TP + FN}$$

- Especificidad. Ratio de verdaderos negativos, es el cociente entre los verdaderos negativos y el total de negativos. La mayoría de tumores benignos (negativo) se clasifiquen como benignos.

$$especificidad = \frac{TN}{TN + FP}$$

```
sens <- TP/(TP+FN)
espec <- TN/(TN+FP)
```

Con el paquete `caret`

```
confusionMatrix(knncv, w_test_label, positive = "M")
```

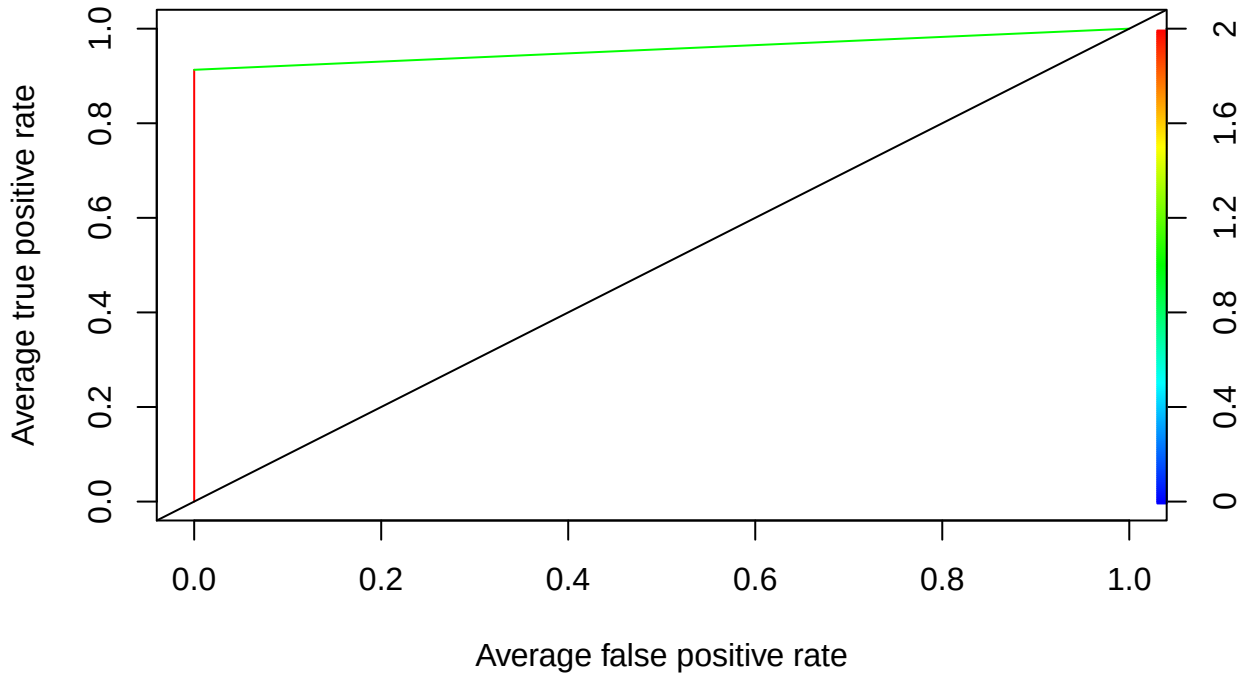
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  B  M
##          B 77  6
##          M  0 17
##
##                Accuracy : 0.94
##                  95% CI : (0.8739701, 0.9776651)
##     No Information Rate : 0.77
##     P-Value [Acc > NIR] : 0.000004732313
##
##                   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
##
```

7.0.4.4. Curvas ROC (*Receiver Operating Characteristic*)

Las curvas ROC (característica operativa del receptor) se desarrollaron durante la Segunda Guerra Mundial como mecanismo para discriminar señales de radio.

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

```
prd<-ifelse(w_predict=="M", 1, 0)
pred_knn<-ROCR::prediction(prd, w_test_label)
perf <- performance(pred_knn, measure = "tpr", x.measure = "fpr")
plot(perf, avg = "threshold", colorize =T)
abline(a=0,b=1)
```




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

```
## [1] "Área bajo la curva:  0.9565"
```