Diagnóstico de cáncer de mama con el algoritmo k-NN

El cribado sistemático del cáncer de mama permite diagnosticar y tratar la enfermedad antes de que produzca síntomas evidentes. El proceso de detección temprana implica examinar el tejido mamario en busca de bultos o masas anormales. Si se encuentra un bulto, se realiza una biopsia por aspiración con aguja fina, que utiliza una aguja hueca para extraer una pequeña muestra de células de la masa. A continuación, un médico examina las células bajo un microscopio para determinar si es probable que la masa sea maligna o benigna.

Si el aprendizaje automático pudiera automatizar la identificación de células cancerosas, proporcionaría un beneficio considerable al sistema de salud. Es probable que los procesos automatizados mejoren la eficiencia del proceso de detección, lo que permitirá a los médicos dedicar menos tiempo al diagnóstico y más tiempo al tratamiento de la enfermedad. Un sistema de cribado automatizado también podría proporcionar una mayor precisión en la detección al eliminar el componente humano inherentemente subjetivo del proceso.

Investigaremos la utilidad del aprendizaje automático para detectar el cáncer aplicando el algoritmo k-NN a las mediciones de células biopsiadas de mujeres con masas mamarias anormales.

Paso 1: recopilación de datos

Utilizaremos el conjunto de datos de diagnóstico de cáncer de mama de Wisconsin del repositorio de aprendizaje automático de la UCI en http://archive.ics.uci.edu/ml. Estos datos fueron donados por investigadores de la Universidad de Wisconsin e incluyen mediciones de imágenes digitalizadas de aspiración con aguja fina de una masa mamaria.

Los valores representan características de los núcleos celulares presentes en la imagen digital.

Los datos de cáncer de mama incluyen 569 ejemplos de biopsias de cáncer, cada una con 32 características. Una característica es un número de identificación, otra es el diagnóstico de cáncer y 30 son mediciones de laboratorio con valores numéricos. El diagnóstico se codifica como "M" para indicar maligno o "B" para indicar benigno.

Las 30 mediciones numéricas comprenden la media, el error estándar y el peor valor (es decir, el mayor) para 10 características diferentes de los núcleos celulares digitalizados. Estos incluyen:

- Radio (Radius)
- Textura (Texture)
- Perímetro (Perimeter)
- Área (Area)
- Suavidad (Smoothness)
- Compacidad (Compactness)
- Concavidad (Concavity)
- Puntos cóncavos (Concave points)
- Simetría (Symmetry)
- Dimensión fractal (Fractal dimension)

Según estos nombres, todas las características parecen estar relacionadas con la forma y el tamaño de los núcleos celulares. A menos que seas un oncólogo, es poco probable que sepas cómo se relaciona cada una con masas benignas o malignas. Estos patrones se revelarán a medida que avancemos en el proceso de aprendizaje automático.

Paso 2: exploración y preparación de los datos

Exploremos los datos y veamos si podemos arrojar algo de luz sobre las relaciones.
Al hacerlo, prepararemos los datos para su uso con el método de aprendizaje k-NN.

Para este ejemplo, el conjunto de datos se modificó muy levemente de su forma original. En particular, se agregó una línea de encabezado y las filas de datos se ordenaron aleatoriamente. Comenzaremos importando el archivo de datos CSV como lo hemos hecho en ejemplos anteriores, guardando los datos de cáncer de mama de Wisconsin en el marco de datos wbcd:

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

Con el comando str(wbcd), podemos confirmar que los datos están estructurados con 569 ejemplos y 32 características, como esperábamos. Las primeras líneas de salida son las siguientes:

```
> str(wbcd)
'data.frame':   569 obs. of  32 variables:
 $ id               : int  87139402 8910251 905520 868871 9012568 906539 925291 87880 862989 89827 ...
 $ diagnosis        : chr  "B" "B" "B" "B" ...
 $ radius_mean      : num  12.3 10.6 11 11.3 15.2 ...
 $ texture_mean     : num  12.4 18.9 16.8 13.4 13.2 ...
 $ perimeter_mean   : num  78.8 69.3 70.9 73 97.7 ...
 $ area_mean        : num  464 346 373 385 712 ...
```

La primera variable es una variable entera llamada id. Como se trata simplemente de un identificador (ID) único para cada paciente en los datos, no proporciona información útil y tendremos que excluirla del modelo.

Independientemente del método de aprendizaje automático, las variables ID siempre deben excluirse. No hacerlo puede llevar a hallazgos erróneos porque el ID se puede utilizar para predecir correctamente cada ejemplo. Por lo tanto, un modelo que incluye una columna ID casi definitivamente sufrirá de sobreajuste y se generalizará mal a datos futuros.

Eliminemos por completo la característica id. Como se encuentra en la primera columna, podemos excluirla haciendo una copia del marco de datos wbcd sin la columna 1:

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

La siguiente variable, diagnóstico, es de particular interés ya que es el resultado que esperamos predecir. Esta característica indica si el ejemplo es de una masa benigna o maligna. La salida de table() indica que 357 masas son benignas, mientras que 212 son malignas:

```
> table(wbcd$diagnosis)

  B   M
357 212
```

Muchos clasificadores de aprendizaje automático de R requieren que la característica de destino se codifique como un factor, por lo que necesitaremos recodificar la variable de diagnóstico. También aprovecharemos esta oportunidad para dar a los valores "B" y "M" etiquetas más informativas mediante el parámetro labels:

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

Cuando observamos la salida de prop.table(), ahora encontramos que los valores se han etiquetado como Benigno y Maligno, con el 62.7 por ciento y el 37.3 por ciento de las masas, respectivamente:

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

   Benign Malignant
     62.7      37.3
```

Las 30 características restantes son todas numéricas y, como se esperaba, consisten en tres mediciones diferentes de 10 características. Con fines ilustrativos, solo analizaremos más de cerca tres de estas características:

```
> summary(wbcd[c("radius_mean", "area_mean", "smoothness_mean")])

  radius_mean       area_mean      smoothness_mean
 Min.   : 6.981   Min.   : 143.5   Min.   :0.05263
 1st Qu.:11.700   1st Qu.: 420.3   1st Qu.:0.08637
 Median :13.370   Median : 551.1   Median :0.09587
 Mean   :14.127   Mean   : 654.9   Mean   :0.09636
 3rd Qu.:15.780   3rd Qu.: 782.7   3rd Qu.:0.10530
 Max.   :28.110   Max.   :2501.0   Max.   :0.16340
```

Al observar las tres una al lado de la otra, ¿notas algo problemático sobre los valores? Recuerda que el cálculo de la distancia para k-NN depende en gran medida de la escala de medición de las características de entrada. Dado que la suavidad varía de 0.05 a 0.16, mientras que el área varía de 143.5 a 2501.0, el impacto del área será mucho mayor que la suavidad en el cálculo de la distancia. Esto podría causar problemas para nuestro clasificador, así que apliquemos la normalización para reescalar las características a un rango estándar de valores.

Transformación: normalización de datos numéricos

Para normalizar estas características, necesitamos crear una función normalize() en R. Esta función toma un vector *x* de valores numéricos y, para cada valor en *x*, resta el valor *x* mínimo y lo divide por el rango de valores *x*. Por último, se devuelve el vector resultante. El código para la función es el siguiente:

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

Después de ejecutar el código anterior, la función normalize() está disponible para su uso en R. Probemos la función en un par de vectores:

```
> normalize(c(1, 2, 3, 4, 5))
[1] 0.00 0.25 0.50 0.75 1.00
> normalize(c(10, 20, 30, 40, 50))
[1] 0.00 0.25 0.50 0.75 1.00
```

La función parece funcionar correctamente. A pesar de que los valores en el segundo vector son 10 veces más grandes que los del primer vector, después de la normalización, ambos parecen exactamente iguales.

Ahora podemos aplicar la función normalize() a las características numéricas de nuestro marco de datos. En lugar de normalizar cada una de las 30 variables numéricas individualmente, utilizaremos una de las funciones de R para automatizar el proceso.

La función lapply() toma una lista y aplica una función específica a cada elemento de la lista. Como un marco de datos es una lista de vectores de igual longitud, podemos utilizar lapply() para aplicar normalize() a cada característica del marco de datos. El paso final es convertir la lista devuelta por lapply() en un marco de datos utilizando la función as.data.frame(). El proceso completo se ve así:

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

En términos sencillos, este comando aplica la función normalize() a las columnas 2 a 31 en el marco de datos wbcd, convierte la lista resultante en un marco de datos y le asigna el nombre wbcd_n. El sufijo _n se utiliza aquí como recordatorio de que los valores en wbcd se han normalizado.

Para confirmar que la transformación se aplicó correctamente, observemos las estadísticas de resumen de una variable:

```
> summary(wbcd_n$area_mean)
   Min.   1st Qu.  Median    Mean   3rd Qu.    Max.
 0.0000  0.1174   0.1729    0.2169  0.2711   1.0000
```

Como se esperaba, la variable area_mean, que originalmente oscilaba entre 143.5 y 2501.0, ahora oscila entre 0 y 1.

Preparación de datos: creación de conjuntos de datos de prueba y entrenamiento

Aunque las 569 biopsias están etiquetadas con un estado benigno o maligno, no es muy interesante predecir lo que ya sabemos. Además, cualquier medida de rendimiento que obtengamos durante el entrenamiento puede ser engañosa, ya que no sabemos hasta qué punto se han sobreajustado los datos ni qué tan bien el aprendiz generalizará a nuevos casos. Por estos motivos, una pregunta más interesante es qué tan bien se desempeña nuestro aprendiz en un conjunto de datos no vistos. Si tuviéramos acceso a un laboratorio, podríamos aplicar nuestro aprendiz a mediciones tomadas de las siguientes 100 masas de estado de cáncer desconocido y ver qué tan bien se comparan las predicciones del aprendiz automático con los diagnósticos obtenidos utilizando métodos convencionales.

En ausencia de dichos datos, podemos simular este escenario dividiendo nuestros datos en dos partes: un conjunto de datos de entrenamiento que se utilizará para construir el modelo

k-NN y un conjunto de datos de prueba que se utilizará para estimar la precisión predictiva del modelo. Utilizaremos los primeros 469 registros para el conjunto de datos de entrenamiento y los 100 restantes para simular nuevos pacientes.

Usando los métodos de extracción de datos presentados previamente en el curso, dividiremos el marco de datos wbcd_n en wbcd_train y wbcd_test:

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

Si los comandos anteriores son confusos, recuerda que los datos se extraen de los marcos de datos utilizando la sintaxis [fila, columna]. Un valor en blanco para el valor de la fila o columna indica que se deben incluir todas las filas o columnas. Por lo tanto, la primera línea de código solicita las filas 1 a 469 y todas las columnas, y la segunda línea solicita 100 filas de la 470 a la 569 y todas las columnas.

> Al construir conjuntos de datos de entrenamiento y prueba, es importante que cada conjunto de datos sea un subconjunto representativo del conjunto completo de datos. Los registros wbcd ya estaban ordenados aleatoriamente, por lo que simplemente podríamos extraer 100 registros consecutivos para crear un conjunto de datos de prueba. Esto no sería apropiado si los datos estuvieran ordenados cronológicamente o en grupos de valores similares. En estos casos, se necesitarían métodos de muestreo aleatorio. El muestreo aleatorio se analizará posteriormente.

Cuando construimos nuestros conjuntos de datos de entrenamiento y prueba normalizados, excluimos la variable objetivo, el diagnóstico (*diagnosis*). Para entrenar el modelo k-NN, necesitaremos almacenar estas etiquetas de clase en vectores de factores, divididos entre los conjuntos de datos de entrenamiento y prueba:

```
> wbcd_train_labels <- wbcd[1:469, 1]
> wbcd_test_labels <- wbcd[470:569, 1]
```

Este código toma el factor de diagnóstico en la primera columna del marco de datos wbcd y crea los vectores wbcd_train_labels y wbcd_test_labels. Los usaremos en los próximos pasos de entrenamiento y evaluación de nuestro clasificador.

Paso 3: entrenar un modelo con los datos

Con nuestros datos de entrenamiento y el vector de etiquetas, ahora estamos listos para clasificar nuestros registros de prueba. Para el algoritmo k-NN, la fase de entrenamiento en realidad no implica la construcción de un modelo; el proceso de entrenamiento de un aprendiz

perezoso como k-NN simplemente implica almacenar los datos de entrada en un formato estructurado.

Para clasificar nuestras instancias de prueba, usaremos una implementación k-NN del paquete class, que proporciona un conjunto de funciones básicas de R para la clasificación. Si este paquete aún no está instalado en tu sistema, puedes instalarlo escribiendo:

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

Para cargar el paquete durante cualquier sesión en la que desee usar las funciones, simplemente ingrese el comando library(class).

La función knn() del paquete class proporciona una implementación clásica y estándar del algoritmo k-NN. Para cada instancia de los datos de prueba, la función identificará los k vecinos más cercanos, utilizando la distancia euclidiana, donde k es un número especificado por el usuario. La instancia de prueba se clasifica mediante una "votación" entre los k vecinos más cercanos; específicamente, esto implica asignar la clase de la mayoría de los vecinos. En caso de empate, la votación se resuelve de forma aleatoria.

Existen otras funciones k-NN en otros paquetes R que proporcionan implementaciones más sofisticadas o más eficientes. Si encuentras limitaciones con knn(), busca k-NN en la Red de Archivos R Integral (CRAN).

El entrenamiento y la clasificación con la función knn() se realizan en un solo comando con cuatro parámetros, como se muestra en la siguiente tabla:

kNN classification syntax

using the `knn()` function in the `class` package

Building the classifier and making predictions:

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

- `train` is a data frame containing numeric training data
- `test` is a data frame containing numeric test data
- `class` is a factor vector with the class for each row in the training data
- `k` is an integer indicating the number of nearest neighbors

The function returns a factor vector of predicted classes for each row in the test data frame.

Example:

```
wbcd_pred <- knn(train = wbcd_train, test = wbcd_test,
                 cl = wbcd_train_labels, k = 3)
```

Ahora tenemos casi todo lo que necesitamos para aplicar el algoritmo k-NN a estos datos. Hemos dividido nuestros datos en conjuntos de datos de entrenamiento y prueba, cada uno

con exactamente las mismas características numéricas. Las etiquetas de los datos de entrenamiento se almacenan en un vector de factores separado. El único parámetro restante es k, que especifica la cantidad de vecinos que se incluirán en la votación.

Como nuestros datos de entrenamiento incluyen 469 instancias, podríamos probar k = 21, un número impar aproximadamente igual a la raíz cuadrada de 469. Con un resultado de dos categorías, el uso de un número impar elimina la posibilidad de terminar con un empate en la votación.

Ahora podemos utilizar la función knn() para clasificar los datos de prueba:

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

La función knn() devuelve un vector de factores de etiquetas predichas para cada uno de los ejemplos en el conjunto de datos wbcd_test. Hemos asignado estas predicciones a wbcd_test_pred.

Paso 4: evaluación del rendimiento del modelo

El siguiente paso del proceso es evaluar qué tan bien coinciden las clases predichas en el vector wbcd_test_pred con los valores reales en el vector wbcd_test_labels. Para ello, podemos utilizar la función CrossTable() en el paquete gmodels, que se presentó previamente.
Si aún no lo has hecho, instala este paquete utilizando el comando:
> install.packages("gmodels").

Después de cargar el paquete con el comando library(gmodels), podemos crear una tabulación cruzada que indique la concordancia entre los vectores de etiquetas predichos y reales. Si se especifica prop.chisq = FALSE, se eliminarán los valores de chi-cuadrado innecesarios de la salida:

```
> CrossTable(x = wbcd_test_labels, y = wbcd_test_pred,
+   prop.chisq = FALSE)
```

Los porcentajes de celdas en la tabla indican la proporción de valores que caen en cuatro categorías. La celda superior izquierda indica los resultados negativos verdaderos. Estos 61 de 100 valores son casos en los que la masa era benigna y el algoritmo k-NN la identificó correctamente como tal. La celda inferior derecha indica los resultados positivos verdaderos, en los que el clasificador y la etiqueta determinada clínicamente coinciden en que la masa es maligna. Un total de 37 de 100 predicciones fueron positivos verdaderos.

La tabla resultante se ve así:

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

Total Observations in Table:  100

                 | wbcd_test_pred
wbcd_test_labels |    Benign | Malignant | Row Total |
-----------------|-----------|-----------|-----------|
          Benign |        61 |         0 |        61 |
                 |     1.000 |     0.000 |     0.610 |
                 |     0.968 |     0.000 |           |
                 |     0.610 |     0.000 |           |
-----------------|-----------|-----------|-----------|
       Malignant |         2 |        37 |        39 |
                 |     0.051 |     0.949 |     0.390 |
                 |     0.032 |     1.000 |           |
                 |     0.020 |     0.370 |           |
-----------------|-----------|-----------|-----------|
    Column Total |        63 |        37 |       100 |
                 |     0.630 |     0.370 |           |
-----------------|-----------|-----------|-----------|
```

Las celdas que caen en la otra diagonal contienen recuentos de ejemplos en los que la predicción k-NN no coincidió con la etiqueta verdadera. Los dos ejemplos en la celda inferior izquierda son resultados negativos falsos; en este caso, el valor predicho era benigno, pero el tumor era en realidad maligno. Los errores en esta dirección podrían ser extremadamente costosos, ya que podrían llevar a un paciente a creer que no tiene cáncer, pero en realidad, la enfermedad puede seguir propagándose. La celda superior derecha contendría los resultados falsos positivos, si los hubiera. Estos valores se producen cuando el modelo ha clasificado una masa como maligna cuando en realidad era benigna. Aunque estos errores son menos peligrosos que un resultado falso negativo, también deben evitarse, ya que podrían suponer una carga financiera adicional para el sistema sanitario o estrés para el paciente, ya que podrían proporcionarle pruebas o tratamientos innecesarios.

Si quisiéramos, podríamos eliminar totalmente los falsos negativos clasificando cada masa como maligna. Obviamente, esta no es una estrategia realista. Aun así, ilustra el hecho de que la predicción implica lograr un equilibrio entre la tasa de falsos positivos y la tasa de falsos negativos. Más adelante trabajaremos con la evaluación del rendimiento del modelo, aprenderás métodos para evaluar la precisión predictiva que se pueden utilizar para optimizar el rendimiento en función de los costos de cada tipo de error.

Un total de 2 de cada 100, o el 2 por ciento de las masas, se clasificaron incorrectamente con el enfoque k-NN. Si bien una precisión del 98 por ciento parece impresionante para unas

pocas líneas de código R, podríamos probar otra iteración del modelo para ver si podemos mejorar el rendimiento y reducir la cantidad de valores que se han clasificado incorrectamente, especialmente porque los errores eran falsos negativos peligrosos.

Paso 5: mejorar el rendimiento del modelo

Intentaremos dos variaciones simples de nuestro clasificador anterior. Primero, emplearemos un método alternativo para reescalar nuestras características numéricas. Segundo, probaremos varios valores k diferentes.

Transformación – Estandarización de la puntuación z

Aunque la normalización se utiliza habitualmente para la clasificación de k-NN, la estandarización de la puntuación z puede ser una forma más adecuada de reescalar las características en un conjunto de datos de cáncer. Dado que los valores estandarizados de la puntuación z no tienen un mínimo ni un máximo predefinidos, los valores extremos no se comprimen hacia el centro. Incluso en ausencia de una formación formal en el ámbito médico, se podría sospechar que un tumor maligno puede dar lugar a valores atípicos extremos a medida que los tumores crecen sin control. Con esto en mente, puede ser razonable permitir que los valores atípicos tengan un mayor peso en el cálculo de la distancia. Veamos si la estandarización de la puntuación z mejora nuestra precisión predictiva.

Para estandarizar un vector, podemos utilizar la función scale() incorporada de R, que, de forma predeterminada, reescala los valores utilizando la estandarización de la puntuación z. La función scale() se puede aplicar directamente a un marco de datos, por lo que no es necesario utilizar la función lapply(). Para crear una versión estandarizada de los datos wbcd con puntuación z, podemos utilizar el siguiente comando:

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

Esto reescala todas las características con la excepción del diagnóstico en la primera columna y almacena el resultado como el marco de datos wbcd_z. El sufijo _z es un recordatorio de que los valores se transformaron con puntuación z.
Para confirmar que la transformación se aplicó correctamente, podemos observar las estadísticas de resumen:

```
> summary(wbcd_z$area_mean)
   Min.   1st Qu.    Median    Mean   3rd Qu.    Max.
-1.4532  -0.6666  -0.2949    0.0000   0.3632   5.2459
```

La media de una variable estandarizada con puntuación z siempre debe ser cero y el rango debe ser bastante compacto. Una puntuación z mayor que 3 o menor que -3 indica un valor extremadamente raro. Al examinar las estadísticas de resumen con estos criterios en mente, la transformación parece haber funcionado.

Como hemos hecho antes, necesitamos dividir los datos transformados con puntuación z en conjuntos de entrenamiento y prueba, y clasificar las instancias de prueba utilizando la función knn().
Luego compararemos las etiquetas predichas con las etiquetas reales utilizando CrossTable():

```
> wbcd_train <- wbcd_z[1:469, ]
> wbcd_test <- wbcd_z[470:569, ]
> wbcd_train_labels <- wbcd[1:469, 1]
> wbcd_test_labels <- wbcd[470:569, 1]
> wbcd_test_pred <- knn(train = wbcd_train, test = wbcd_test,
+   cl = wbcd_train_labels, k = 21)
> CrossTable(x = wbcd_test_labels, y = wbcd_test_pred,
+   prop.chisq = FALSE)
```

Desafortunadamente, en la siguiente tabla, los resultados de nuestra nueva transformación muestran una leve disminución en la precisión. Utilizando los mismos casos en los que antes habíamos clasificado correctamente el 98 por ciento de los ejemplos, ahora clasificamos correctamente sólo el 95 por ciento. Para empeorar las cosas, no clasificamos mejor los peligrosos falsos negativos.

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


Total Observations in Table:  100


                 | wbcd_test_pred
wbcd_test_labels |    Benign | Malignant | Row Total |
-----------------|-----------|-----------|-----------|
          Benign |        61 |         0 |        61 |
                 |     1.000 |     0.000 |     0.610 |
                 |     0.924 |     0.000 |           |
                 |     0.610 |     0.000 |           |
-----------------|-----------|-----------|-----------|
       Malignant |         5 |        34 |        39 |
                 |     0.128 |     0.872 |     0.390 |
                 |     0.076 |     1.000 |           |
                 |     0.050 |     0.340 |           |
-----------------|-----------|-----------|-----------|
    Column Total |        66 |        34 |       100 |
                 |     0.660 |     0.340 |           |
-----------------|-----------|-----------|-----------|
```

Prueba de valores alternativos de k

Es posible que podamos optimizar el rendimiento del modelo k-NN examinando su rendimiento en varios valores k. Utilizando los conjuntos de datos de prueba y de entrenamiento normalizados, se clasificaron los mismos 100 registros utilizando varias opciones diferentes de k. La cantidad de falsos negativos y falsos positivos se muestra para cada iteración:

Valor k	Falsos negativos	Falsos positivos	Porcentaje clasificado incorrectamente
1	1	3	4%
5	2	0	2%
11	3	0	3%
15	3	0	3%
21	2	0	2%
27	4	0	4%

Aunque el clasificador nunca fue perfecto, el enfoque de 1-NN pudo evitar algunos de los falsos negativos a expensas de agregar falsos positivos. Sin embargo, es importante tener en cuenta que no sería prudente adaptar demasiado nuestro enfoque a nuestros datos de prueba; después de todo, es probable que un conjunto diferente de 100 registros de pacientes sea algo diferente de los utilizados para medir nuestro rendimiento. El estudiante deberá comprobar esta tabla, analizando los resultados y comprobar.

Si necesitas estar seguro de que un alumno generalizará a datos futuros, puedes crear varios conjuntos de 100 pacientes al azar y volver a probar el resultado repetidamente. Estos métodos para evaluar cuidadosamente el rendimiento de los modelos de aprendizaje automático se analizarán más a fondo en temas futuros, evaluación del rendimiento del modelo.

Resumen

En este ejemplo, aprendimos sobre la clasificación mediante k-NN. A diferencia de muchos algoritmos de clasificación, k-nearest neighbors no realiza ningún aprendizaje. Simplemente almacena los datos de entrenamiento textualmente. Los ejemplos de prueba sin etiquetar se combinan luego con los registros más similares en el conjunto de entrenamiento utilizando una función de distancia, y al ejemplo sin etiquetar se le asigna la etiqueta de sus vecinos.

A pesar del hecho de que k-NN es un algoritmo muy simple, es capaz de abordar tareas extremadamente complejas como la identificación de masas cancerosas. En unas pocas líneas simples de código R, pudimos identificar correctamente si una masa era maligna o benigna el 98 por ciento de las veces.

En el próximo tema, examinaremos un método de clasificación que utiliza la probabilidad para estimar la probabilidad de que una observación caiga en ciertas categorías. Será interesante comparar en qué se diferencia este enfoque de k-NN. Más adelante, en temas futuros del curso, clustering, aprenderemos sobre un pariente cercano de k-NN, que utiliza medidas de distancia para una tarea de aprendizaje completamente diferente.