Evaluación del rendimiento del modelo, parte II

Estimación del rendimiento futuro

Algunos paquetes de aprendizaje automático de R presentan matrices de confusión y medidas de rendimiento durante el proceso de creación del modelo. El propósito de estas estadísticas es proporcionar información sobre el **error de resustitución** del modelo, que se produce cuando los datos de entrenamiento se predicen incorrectamente a pesar de que el modelo se crea directamente a partir de estos datos.

Esta información se puede utilizar como un diagnóstico aproximado para identificar modelos con un rendimiento obviamente deficiente.

Un modelo que no puede funcionar lo suficientemente bien con los datos con los que fue entrenado probablemente no lo hará bien con datos futuros.

Lo contrario no es cierto. En otras palabras, no se puede suponer que un modelo que funciona bien con los datos de entrenamiento funcione bien con conjuntos de datos futuros. Por ejemplo, un modelo que utilizara la memorización mecánica para clasificar perfectamente cada instancia de entrenamiento con un error de resustitución cero no podría generalizar sus predicciones a datos que nunca había visto. Por este motivo, se puede suponer que la tasa de error en los datos de entrenamiento es optimista sobre el rendimiento futuro de un modelo.

En lugar de confiar en el error de resustitución, una mejor práctica es evaluar el rendimiento de un modelo con datos que aún no ha visto. Usamos este método en temas anteriores cuando dividimos los datos disponibles en un conjunto para entrenamiento y un conjunto para prueba. Sin embargo, en algunos casos, no siempre es ideal crear conjuntos de datos de entrenamiento y prueba.

Por ejemplo, en una situación en la que solo tiene un pequeño grupo de datos, es posible que no desee reducir más la muestra.
Afortunadamente, como pronto aprenderás, existen otras formas de estimar el rendimiento de un modelo con datos no vistos.

El paquete caret que usamos para calcular las medidas de rendimiento también ofrece funciones para estimar el rendimiento futuro. Si estás siguiendo los ejemplos de código R y aún no has instalado el paquete caret, hazlo. También deberás cargar el paquete en la sesión R mediante el comando library(caret).

El método de retención (holdout)

El procedimiento de partición de datos en conjuntos de datos de entrenamiento y prueba que usamos en temas anteriores se conoce como el **método de retención**. Como se muestra en la figura 15, el **conjunto de datos de entrenamiento** se usa para generar el modelo, que luego se aplica al **conjunto de datos de prueba** para generar predicciones para la evaluación.

Normalmente, aproximadamente un tercio de los datos se reserva para la prueba y dos tercios se usan para el entrenamiento, pero esta proporción puede variar según la cantidad de datos disponibles o la complejidad de la tarea de aprendizaje. Para garantizar que los conjuntos de datos de entrenamiento y prueba no tengan diferencias sistemáticas, sus ejemplos se dividen aleatoriamente en dos grupos.

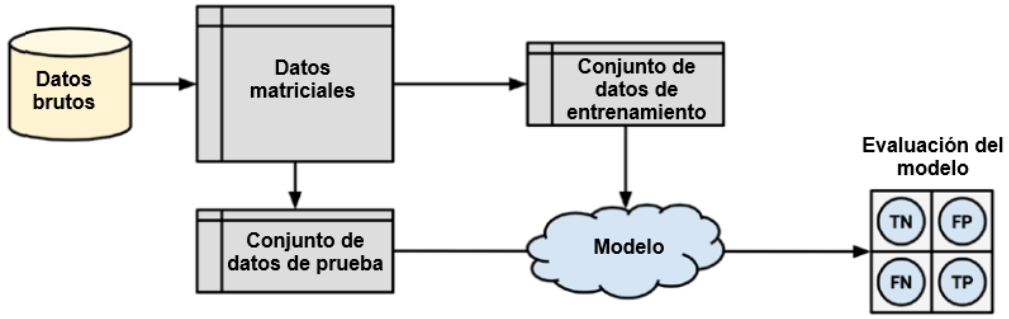

Figura 15: El método de retención más simple divide los datos en conjuntos de entrenamiento y de prueba.

Para que el método de retención dé como resultado una estimación verdaderamente precisa del rendimiento futuro, en ningún momento se debe permitir que el rendimiento del conjunto de datos de prueba influya en el proceso de modelado.

En palabras del profesor de Stanford y reconocido experto en aprendizaje automático Trevor Hastie, "*lo ideal es que el conjunto de prueba se mantenga en una 'bóveda' y se saque solo al final del análisis de datos*". En otras palabras, los datos de prueba deben permanecer intactos más allá de su único propósito, que es evaluar un único modelo final.

Es fácil violar esta regla sin saberlo y echar un vistazo a la "bóveda" metafórica al elegir uno de varios modelos o cambiar un solo modelo en función de los resultados de pruebas repetidas. Por ejemplo, supongamos que creamos varios modelos con los datos de entrenamiento y seleccionamos el que tiene la mayor precisión con los datos de prueba. En este caso, debido a que hemos utilizado el conjunto de datos de prueba para seleccionar el mejor resultado, el rendimiento de la prueba no es una medida imparcial del rendimiento futuro con datos no vistos.

Un lector atento notará que los datos de prueba de reserva se utilizaron en temas anteriores tanto para evaluar modelos como para mejorar el rendimiento del modelo. Esto se hizo con fines ilustrativos, pero de hecho violaría la regla establecida anteriormente. En consecuencia, las estadísticas de rendimiento del modelo que se muestran no fueron estimaciones verdaderamente imparciales del rendimiento futuro con datos no vistos.

Para evitar este problema, es mejor dividir los datos originales de modo que, además de los conjuntos de datos de entrenamiento y prueba, esté disponible un **conjunto de datos de validación**. El conjunto de datos de validación se puede utilizar para iterar y refinar el modelo o los modelos elegidos, dejando el conjunto de datos de prueba para utilizarlo solo una vez como paso final para informar una tasa de error estimada para predicciones futuras. Una división típica entre entrenamiento, prueba y validación sería 50 por ciento, 25 por ciento y 25 por ciento, respectivamente.

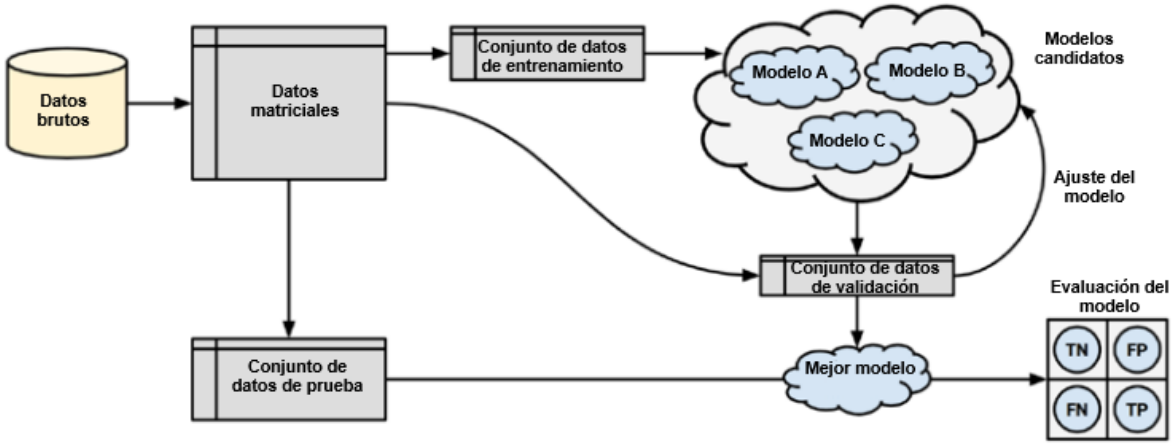

Figura 16: Un conjunto de datos de validación se puede dejar de lado del entrenamiento para seleccionar entre varios modelos candidatos.

Un método simple para crear muestras de reserva utiliza generadores de números aleatorios para asignar registros a particiones. Esta técnica se utilizó por primera vez en el tema, Divide and Conquista – Clasificación mediante árboles de decisión y reglas, para crear conjuntos de datos de entrenamiento y prueba.

Si deseas seguir los siguientes ejemplos, descarga el conjunto de datos credit.csv y cárgalo en un frame de datos utilizando el comando credit <- read.csv("credit.csv", stringsAsFactors = TRUE).

Supongamos que tenemos un frame de datos llamado credit con 1000 filas de datos. Podemos dividirlo en tres particiones de la siguiente manera. Primero, creamos un vector de identificadores de fila ordenados aleatoriamente de 1 a 1000 utilizando la función runif(), que, de manera predeterminada, genera una cantidad específica de valores aleatorios entre 0 y 1. La función runif() recibe su nombre de la distribución uniforme aleatoria, que se analizó en el documento, Gestión y comprensión de los datos.

Luego, la función order() devuelve un vector que indica el orden de clasificación de los 1000 números aleatorios. Por ejemplo, order(c(0.5, 0.25, 0.75, 0.1)) devuelve la secuencia 4 2 1 3 porque el número más pequeño (0.1) aparece en cuarto lugar, el segundo más pequeño (0.25) aparece en segundo lugar, y así sucesivamente:

```
> random_ids <- order(runif(1000))
```

A continuación, los identificadores aleatorios se utilizan para dividir el frame de datos de crédito en 500, 250 y 250 registros que comprenden el conjunto de datos de entrenamiento, validación y prueba.

```
> credit_train <- credit[random_ids[1:500], ]
> credit_validate <- credit[random_ids[501:750], ]
> credit_test <- credit[random_ids[751:1000], ]
```

Un problema con el muestreo de retención es que cada partición puede tener una proporción mayor o menor de algunas clases. En los casos en que una (o más) clase es una proporción muy pequeña del conjunto de datos, esto puede provocar que se omita del conjunto de datos de entrenamiento, un problema importante porque el modelo no puede aprender esta clase.

Para reducir la posibilidad de que esto ocurra, se puede utilizar una técnica llamada **muestreo aleatorio estratificado**.
Aunque una muestra aleatoria generalmente debe contener aproximadamente la misma proporción de cada valor de clase que el conjunto de datos completo, el muestreo aleatorio estratificado garantiza que las particiones aleatorias tengan casi la misma proporción de cada clase que el conjunto de datos completo, incluso cuando algunas clases son pequeñas.

El paquete caret proporciona una función createDataPartition(), que crea particiones basadas en el muestreo de retención estratificado. Los pasos para crear una muestra estratificada de datos de entrenamiento y prueba para el conjunto de datos de crédito se muestran en los siguientes comandos. Para utilizar la función, se debe especificar un vector de valores de clase (aquí, default se refiere a si un préstamo entró en mora), además de un parámetro, p, que especifica la proporción de instancias que se incluirán en la partición. El parámetro list = FALSE evita que el resultado se almacene como un objeto de lista, una capacidad que es necesaria para técnicas de muestreo más complejas, pero que no es necesaria aquí:

```
> in_train <- createDataPartition(credit$default, p = 0.75, list = FALSE)
> credit_train <- credit[in_train, ]
> credit_test <- credit[-in_train, ]
```

El vector in_train indica los números de fila incluidos en la muestra de entrenamiento. Podemos utilizar estos números de fila para seleccionar ejemplos para el frame de datos credit_train. De manera similar, al utilizar un símbolo negativo, podemos utilizar las filas que no se encuentran en el vector in_train para el conjunto de datos credit_test.

Aunque distribuye las clases de manera uniforme, el muestreo estratificado no garantiza otros tipos de representatividad. Algunas muestras pueden tener demasiados o muy pocos casos difíciles, casos fáciles de predecir o valores atípicos. Esto es especialmente cierto para conjuntos de datos más pequeños, que pueden no tener una porción lo suficientemente grande de dichos casos para dividirlos entre los conjuntos de entrenamiento y prueba.

Además de las muestras potencialmente sesgadas, otro problema con el método de retención es que se deben reservar porciones sustanciales de datos para probar y validar el modelo. Dado que estos datos no se pueden usar para entrenar el modelo hasta que se haya medido su rendimiento, es probable que las estimaciones de rendimiento sean demasiado conservadoras.

Dado que los modelos entrenados en conjuntos de datos más grandes generalmente funcionan mejor, una práctica común es volver a entrenar el modelo en el conjunto completo de datos (es decir, entrenamiento más prueba y validación) después de que se haya seleccionado y evaluado un modelo final.

A veces se utiliza una técnica llamada **retención repetida** para mitigar los problemas de los conjuntos de datos de entrenamiento compuestos aleatoriamente. El método de retención repetida es un caso especial del método de retención que utiliza el resultado promedio de varias muestras de retención aleatorias para evaluar el rendimiento de un modelo. Como se utilizan múltiples muestras de retención, es menos probable que el modelo se entrene o pruebe en datos no representativos. Ampliaremos esta idea en la siguiente sección.

Validación cruzada

La retención repetida es la base de una técnica conocida como **validación cruzada (VC) de k-fold** (k-fold CV, *k-fold cross-validation*), que se ha convertido en el estándar de la industria para estimar el rendimiento del modelo. En lugar de tomar muestras aleatorias repetidas que podrían potencialmente usar el mismo registro más de una vez, la VC de k-fold divide aleatoriamente los datos en *k* particiones aleatorias completamente separadas llamadas pliegues (**folds**).

Aunque *k* se puede establecer en cualquier número, la convención más común es usar VC de 10 pliegues. ¿Por qué 10 pliegues? La razón es que la evidencia empírica sugiere que hay poco beneficio adicional en usar un número mayor. Para cada uno de los 10 pliegues (cada

uno de los cuales comprende el 10 por ciento de los datos totales), se construye un modelo de aprendizaje automático sobre el 90 por ciento restante de los datos.

Luego, la muestra del 10 por ciento del pliegue se usa para la evaluación del modelo. Después de que el proceso de entrenamiento y evaluación del modelo se haya realizado 10 veces (con 10 combinaciones diferentes de entrenamiento/prueba), se informa el rendimiento promedio en todos los pliegues.

Un caso extremo de VC de k-fold es el **método leave-one-out**, que realiza VC de k-fold utilizando un fold para cada uno de los ejemplos de datos. Esto garantiza que se utilice la mayor cantidad de datos para entrenar el modelo. Aunque esto puede parecer útil, es tan costoso computacionalmente que rara vez se utiliza en la práctica.

Los conjuntos de datos para validación cruzada se pueden crear utilizando la función createFolds() en el paquete caret. De manera similar al muestreo aleatorio estratificado, esta función intentará mantener el mismo equilibrio de clases en cada uno de los folds que en el conjunto de datos original. El siguiente es el comando para crear 10 folds, utilizando set.seed(123) para garantizar que los resultados sean reproducibles:

```
> set.seed(123)
> folds <- createFolds(credit$default, k = 10)
```

El resultado de la función createFolds() es una lista de vectores que almacenan los números de fila para cada uno de los $k = 10$ folds solicitados. Podemos echar un vistazo al contenido utilizando str():

```
> str(folds)
```

```
List of 10
 $ Fold01: int [1:100] 14 23 32 42 51 56 65 66 77 95 ...
 $ Fold02: int [1:100] 21 36 52 55 96 115 123 129 162 169 ...
 $ Fold03: int [1:100] 3 22 30 34 37 39 43 58 70 85 ...
 $ Fold04: int [1:100] 12 15 17 18 19 31 40 45 47 57 ...
 $ Fold05: int [1:100] 1 5 7 20 26 35 46 54 106 109 ...
 $ Fold06: int [1:100] 6 27 29 48 68 69 72 73 74 75 ...
 $ Fold07: int [1:100] 10 38 49 60 61 63 88 94 104 108 ...
 $ Fold08: int [1:100] 8 11 24 53 71 76 89 90 91 101 ...
 $ Fold09: int [1:100] 2 4 9 13 16 25 28 44 62 64 ...
 $ Fold10: int [1:100] 33 41 50 67 81 82 100 105 107 118 ...
```

Aquí, vemos que el primer fold se llama Fold01 y almacena 100 números enteros que indican las 100 filas en el frame de datos de crédito para el primer fold. Para crear conjuntos de datos de prueba y entrenamiento para construir y evaluar un modelo, se necesita un paso adicional. Los siguientes comandos muestran cómo crear datos para el primer pliegue. Asignaremos el

10 por ciento seleccionado al conjunto de datos de prueba y usaremos el símbolo negativo para asignar el 90 por ciento restante al conjunto de datos de entrenamiento:

```
> credit01_test <- credit[folds$Fold01, ]
> credit01_train <- credit[-folds$Fold01, ]
```

Para realizar el VC de 10 pliegues completo, este paso se debe repetir un total de 10 veces, primero construyendo un modelo y luego calculando el rendimiento del modelo cada vez. Al final, se promediarán las medidas de rendimiento para obtener el rendimiento general. Afortunadamente, podemos automatizar esta tarea aplicando varias de las técnicas que aprendimos anteriormente.

Para demostrar el proceso, estimaremos la estadística kappa para un modelo de árbol de decisión C5.0 de los datos de crédito utilizando VC de 10 pliegues. Primero, necesitamos cargar algunos paquetes de R:

```
> library(caret)
> library(C50)
> library(irr)
```

caret (para crear los pliegues), C50 (para construir el árbol de decisión) e irr (para calcular kappa). Los dos últimos paquetes se eligieron con fines ilustrativos; Si lo deseas, puedes utilizar un modelo diferente o una medida de rendimiento diferente con la misma serie de pasos.

A continuación, crearemos una lista de 10 pliegues como lo hicimos anteriormente. La función set.seed() se utiliza aquí para garantizar que los resultados sean consistentes si se vuelve a ejecutar el mismo código:

```
> set.seed(123)
> folds <- createFolds(credit$default, k = 10)
```

Por último, aplicaremos una serie de pasos idénticos a la lista de pliegues utilizando la función lapply(). Como se muestra en el código siguiente, debido a que no existe una función que haga exactamente lo que necesitamos, debemos definir nuestra propia función para pasar a lapply().

Nuestra función personalizada divide el frame de datos de crédito en datos de entrenamiento y de prueba, crea un árbol de decisiones utilizando la función C5.0() en los datos de entrenamiento, genera un conjunto de predicciones a partir de los datos de prueba y compara los valores predichos y reales utilizando la función kappa2():

```
> cv_results <- lapply(folds, function(x) {
+   credit_train <- credit[-x, ]
+   credit_test <- credit[x, ]
+   credit_model <- C5.0(default ~ ., data = credit_train)
+   credit_pred <- predict(credit_model, credit_test)
+   credit_actual <- credit_test$default
+   kappa <- kappa2(data.frame(credit_actual, credit_pred))$value
+   return(kappa)
+ })
```

Las estadísticas kappa resultantes se compilan en una lista almacenada en el objeto cv_results, que podemos examinar utilizando str():

```
> str(cv_results)

List of 10
 $ Fold01: num 0.381
 $ Fold02: num 0.525
 $ Fold03: num 0.247
 $ Fold04: num 0.316
 $ Fold05: num 0.387
 $ Fold06: num 0.368
 $ Fold07: num 0.122
 $ Fold08: num 0.141
 $ Fold09: num 0.0691
 $ Fold10: num 0.381
```

Solo queda un paso más en el proceso VC de 10 pliegues: debemos calcular el promedio de estos 10 valores. Aunque te sentirás tentado a escribir mean(cv_results), debido a que cv_results no es un vector numérico, el resultado sería un error. En su lugar, utiliza la función unlist(), que elimina la estructura de lista y reduce cv_results a un vector numérico. A partir de ahí, podemos calcular kappa medio como se esperaba:

```
> mean(unlist(cv_results))
[1] 0.2939567
```

Esta estadística kappa es bastante baja, lo que corresponde a "justo (*fair*)" en la escala de interpretación, lo que sugiere que el modelo de calificación crediticia funciona solo marginalmente mejor que el azar. Más adelante, examinaremos métodos automatizados basados en VC de 10-folds que pueden ayudarnos a mejorar el rendimiento de este modelo.

Dado que el VC proporciona una estimación del rendimiento a partir de múltiples conjuntos de pruebas, también podemos calcular la variabilidad en la estimación. Por ejemplo, la desviación estándar de las 10 iteraciones se puede calcular como:

```
> sd(unlist(cv_results))
[1] 0.1448565
```

Después de encontrar el promedio y la desviación estándar de la métrica de rendimiento, es posible calcular un intervalo de confianza o determinar si dos modelos tienen una diferencia estadísticamente significativa en el rendimiento, lo que significa que es probable que la diferencia sea real y no se deba a una variación aleatoria.

Desafortunadamente, investigaciones recientes han demostrado que el VC viola los supuestos de dichas pruebas estadísticas, en particular la necesidad de que los datos se extraigan de muestras aleatorias independientes, lo que claramente no es el caso de los pliegues de VC, que están vinculados entre sí por definición.

Para una discusión de las limitaciones de las estimaciones de rendimiento tomadas del VC de 10 pliegues, consulta: Cross-validation: what does it estimate and how well does it do it?, Bates S, Hastie T y Tibshirani R, 2022.

Se han desarrollado variantes más sofisticadas de VC para mejorar la solidez de las estimaciones del rendimiento del modelo. Una de esas técnicas es la **VC repetida de k-fold**, que implica aplicar repetidamente la VC de k-fold y promediar los resultados. Una estrategia común es realizar la VC de 10-folds veces.

Aunque es un proceso computacionalmente intensivo, proporciona una estimación del rendimiento incluso más sólida que una VC de 10-folds estándar, ya que el rendimiento se promedia a lo largo de muchos más ensayos. Sin embargo, también viola los supuestos estadísticos y, por lo tanto, las pruebas estadísticas realizadas sobre los resultados pueden estar ligeramente sesgadas.

Quizás el estándar de oro actual para estimar el rendimiento del modelo sea la **validación cruzada anidada** (*nested cross-validation*), que literalmente realiza una VC de k-fold dentro de otro proceso de VC de k-fold. Esta técnica se describe más adelante, y no solo es extremadamente costosa computacionalmente, sino que también es más difícil de implementar e interpretar.

La ventaja de la VC de k-fold anidada es que produce comparaciones verdaderamente válidas del rendimiento del modelo en comparación con la VC de k-fold estándar, que está sesgada debido a que viola los supuestos estadísticos. Por otra parte, el sesgo causado por esta cuestión parece ser menos importante para conjuntos de datos muy grandes, por lo que todavía puede ser razonable (y sigue siendo una práctica común) utilizar los intervalos de confianza o las pruebas de significación derivadas del enfoque VC más simple para ayudar a identificar el “mejor” modelo.

Muestreo bootstrap

Una alternativa un poco menos popular pero aún bastante utilizada al VC k-fold se conoce como **muestreo bootstrap**, **bootstrap** o **bootstrapping** para abreviar. En términos generales, se refieren a métodos estadísticos que utilizan muestras aleatorias de datos para estimar propiedades de un conjunto más grande. Cuando este principio se aplica al rendimiento del modelo de aprendizaje automático, implica la creación de varios conjuntos de datos de prueba y entrenamiento seleccionados aleatoriamente, que luego se utilizan para estimar las estadísticas de rendimiento. Luego, los resultados de los diversos conjuntos de datos aleatorios se promedian para obtener una estimación final del rendimiento futuro.

Entonces, ¿en qué se diferencia este procedimiento del VC k-fold? Mientras que la validación cruzada divide los datos en particiones separadas en las que cada ejemplo puede aparecer solo una vez, el bootstrap permite seleccionar ejemplos varias veces a través de un proceso de **muestreo con reemplazo**. Esto significa que a partir del conjunto de datos original de *n* ejemplos, el procedimiento bootstrap creará uno o más conjuntos de datos de entrenamiento nuevos que también contienen *n* ejemplos, algunos de los cuales se repiten.

Los conjuntos de datos de prueba correspondientes se construyen a partir del conjunto de ejemplos que no se seleccionaron para los respectivos conjuntos de datos de entrenamiento.

En un conjunto de datos bootstrap, la probabilidad de que cualquier instancia dada sea excluida del conjunto de datos de entrenamiento es del 36.8 por ciento. Podemos demostrarlo matemáticamente reconociendo que cada ejemplo tiene una probabilidad de 1/*n* de ser muestreado cada vez que se agrega una de las *n* filas al conjunto de datos de entrenamiento.

Por lo tanto, para estar en el conjunto de prueba, un ejemplo no debe seleccionarse *n* veces. Como la probabilidad de ser elegido es 1/*n*, la probabilidad de no ser elegido es, por lo tanto, 1 - 1/*n*, y la probabilidad de no ser seleccionado *n* veces es la siguiente:

$$\left(1 - \frac{1}{n}\right)^n$$

Usando esta fórmula, si el conjunto de datos que se va a utilizar para el bootstrap contiene 1000 filas, la probabilidad de que un registro aleatorio no sea seleccionado es:

```
> (1 - (1/1000))^1000
[1] 0.3676954
```

De manera similar, para un conjunto de datos con 100 000 filas:

```
> (1 - (1/100000))^100000
[1] 0.3678776
```

Y a medida que n se acerca al infinito, la fórmula se reduce a 1/e, como se muestra aquí:

```
> 1 / exp(1)
[1] 0.3678794
```

Dado que la probabilidad de no ser seleccionado es del 36.8 por ciento, la probabilidad de que se seleccione cualquier instancia para el conjunto de datos de entrenamiento es 100 – 36.8 = 63.2 por ciento. En otras palabras, los datos de entrenamiento representan solo el 63.2 por ciento de los ejemplos disponibles, algunos de los cuales se repiten. A diferencia del VC de 10-fold, que utiliza el 90 por ciento de los ejemplos para el entrenamiento, la muestra bootstrap es menos representativa del conjunto de datos completo.

Debido a que un modelo entrenado con solo el 63.2 por ciento de los datos de entrenamiento probablemente tenga un peor desempeño que un modelo entrenado con un conjunto de entrenamiento más grande, las estimaciones de desempeño del bootstrap pueden ser sustancialmente menores que las que se obtendrán cuando el modelo se entrene más tarde con el conjunto de datos completo.

Un caso especial de bootstrap, conocido como **bootstrap 0.632**, explica esto al calcular la medida de desempeño final como una función del desempeño tanto en los datos de entrenamiento (que es demasiado optimista) como en los datos de prueba (que es demasiado pesimista). La tasa de error final se estima entonces como:

$$\text{error} = 0.632 \times \text{error}_{\text{test}} + 0.368 \times \text{error}_{\text{train}}$$

Una ventaja del muestreo bootstrap sobre el VC es que tiende a funcionar mejor con conjuntos de datos muy pequeños. Además, el muestreo bootstrap tiene aplicaciones más allá de la medición de desempeño.

En particular, podríamos como se verá posteriormente en un tema, Cómo desarrollar mejores aprendices, aprenderás cómo se pueden usar los principios del muestreo bootstrap para mejorar el desempeño del modelo.

Resumen

En este documento se presentaron varias de las medidas y técnicas más comunes para evaluar el rendimiento de los modelos de clasificación de aprendizaje automático. Aunque la precisión proporciona un método simple para examinar la frecuencia con la que un modelo es correcto, esto puede ser engañoso en el caso de eventos raros porque el costo real de tales eventos puede ser inversamente proporcional a la frecuencia con la que aparecen en los datos.

Algunas medidas basadas en matrices de confusión capturan mejor el desempeño de un modelo, así como el equilibrio entre los costos de varios tipos de errores. La estadística kappa y el coeficiente de correlación de Matthews son dos medidas de desempeño más sofisticadas, que funcionan bien incluso para conjuntos de datos severamente desequilibrados. Además, examinar de cerca las compensaciones entre sensibilidad y especificidad, o precisión y recuperación, puede ser una herramienta útil para pensar en las implicaciones de los errores en el mundo real. Las visualizaciones como la curva ROC también son útiles para este fin.

También vale la pena mencionar que, a veces, la mejor medida del desempeño de un modelo es considerar qué tan bien cumple, o no cumple, otros objetivos. Por ejemplo, puede ser necesario explicar la lógica de un modelo en un lenguaje simple, lo que eliminaría algunos modelos de la consideración. Además, incluso si funciona muy bien, un modelo que es demasiado lento o difícil de escalar a un entorno de producción es completamente inútil.

De cara a los siguientes temas, una extensión obvia de la medición del desempeño es encontrar formas de mejorarlo. A medida que avances en el curso, aplicarás muchos de los principios de este documento mientras fortaleces tus habilidades de aprendizaje automático y agregas habilidades más avanzadas.

Las técnicas de VC, las curvas ROC, el bootstrap y el paquete caret volverán a aparecer regularmente en las próximas páginas, a medida que construimos sobre nuestro trabajo hasta ahora para investigar formas de crear modelos más inteligentes mediante la iteración, el refinamiento y la combinación sistemática de algoritmos de aprendizaje.