
Evaluación del rendimiento del modelo, parte I

Cuando sólo los ricos podían permitirse la educación, los exámenes no evaluaban a los estudiantes, sino que los profesores eran evaluados por los padres que querían saber si sus hijos aprendían lo suficiente para justificar el salario de los instructores. Obviamente, hoy esto es diferente.

Ahora, estas evaluaciones se utilizan para distinguir entre los estudiantes de alto rendimiento y los de bajo rendimiento, y filtrarlos para carreras y otras oportunidades.

Dada la importancia de este proceso, se invierte mucho esfuerzo en desarrollar evaluaciones precisas de los estudiantes. Las evaluaciones justas tienen una gran cantidad de preguntas que cubren una amplia gama de temas y premian el conocimiento verdadero por sobre las conjeturas afortunadas.

Una buena evaluación también requiere que los estudiantes piensen en problemas que nunca han enfrentado antes. Por lo tanto, las respuestas correctas reflejan una capacidad para generalizar el conocimiento de manera más amplia.

El proceso de evaluación de algoritmos de aprendizaje automático es muy similar al proceso de evaluación de estudiantes. Dado que los algoritmos tienen diferentes fortalezas y debilidades, las pruebas deben distinguir entre los aprendices. También es importante comprender cómo se desempeñará un aprendiz en datos futuros.

Este documento proporciona la información necesaria para evaluar a los aprendices automáticos, como:

- Las razones por las que la precisión predictiva no es suficiente para medir el rendimiento y las medidas de rendimiento que puedes utilizar en su lugar.
- Métodos para garantizar que las medidas de rendimiento reflejen razonablemente la capacidad de un modelo para predecir o pronosticar casos no vistos.
- Cómo utilizar R para aplicar estas medidas y métodos más útiles a los modelos predictivos tratados en los temas anteriores.

Así como la mejor manera de aprender un tema es intentar enseñárselo a otra persona, el proceso de enseñar y evaluar a los aprendices automáticos te proporcionará una mayor comprensión de los métodos que has aprendido hasta ahora.

Medición del rendimiento para la clasificación

En los temas anteriores, medimos la precisión del clasificador dividiendo la cantidad de predicciones correctas por la cantidad total de predicciones. Esto determina la proporción de casos en los que el aprendiz tiene razón o no. Por ejemplo, supongamos que un clasificador predijo correctamente para 99,990 de 100,000 bebés recién nacidos si eran portadores de un defecto genético tratable pero potencialmente fatal. Esto implicaría una precisión del 99.99 por ciento y una tasa de error de sólo el 0.01 por ciento.

A primera vista, parece ser un clasificador extremadamente valioso. Sin embargo, sería prudente recopilar información adicional antes de confiar la vida de un niño a la prueba.

¿Qué pasa si el defecto genético se encuentra sólo en 10 de cada 100,000 bebés? Una prueba que invariablemente predice que no hay defecto será correcta para el 99.99 por ciento de todos los casos, pero incorrecta para el 100 por ciento de los casos que más importan. En otras palabras, aunque el clasificador es extremadamente preciso, no es muy útil para prevenir defectos de nacimiento tratables.

Esta es una consecuencia del **problema del desequilibrio de clases**, que se refiere al problema asociado con los datos que tienen una gran mayoría de registros que pertenecen a una sola clase.

Aunque hay muchas formas de medir el rendimiento de un clasificador, la mejor medida es siempre la que captura si el clasificador es exitoso en su propósito previsto. Es crucial definir medidas de rendimiento para la utilidad en lugar de la precisión bruta. Para ello, exploraremos una variedad de medidas de rendimiento alternativas derivadas de la matriz de confusión. Sin embargo, antes de comenzar, debemos considerar cómo preparar un clasificador para la evaluación.

Comprender las predicciones de un clasificador

El objetivo de evaluar un modelo de clasificación es comprender mejor cómo se extrapolará su rendimiento a casos futuros. Dado que normalmente no es factible probar un modelo no probado en un entorno real, normalmente simulamos condiciones futuras pidiendo al modelo que clasifique casos en un conjunto de datos compuesto por casos que se asemejen a lo que se le pedirá que haga en el futuro. Al observar las respuestas del aprendiz a este examen, podemos conocer sus fortalezas y debilidades.

Aunque hemos evaluado los clasificadores en capítulos anteriores, vale la pena reflexionar sobre los tipos de datos que tenemos a nuestra disposición:

- Valores de clase reales
- Valores de clase previstos

- Probabilidad estimada de la predicción

Los valores de clase reales y previstos pueden ser evidentes, pero son la clave de la evaluación. Al igual que un profesor utiliza una clave de respuestas (una lista de respuestas correctas) para evaluar las respuestas del estudiante, necesitamos saber la respuesta correcta para las predicciones de un aprendiz automático. El objetivo es mantener dos vectores de datos: uno que contenga los valores de clase correctos o reales, y el otro que contenga los valores de clase previstos.

Ambos vectores deben tener la misma cantidad de valores almacenados en el mismo orden. Los valores previstos y reales pueden almacenarse como vectores R separados o como columnas en un único frame de datos R .

Obtener estos datos es fácil. Los valores de clase reales provienen directamente del objetivo en el conjunto de datos de prueba. Los valores de clase previstos se obtienen del clasificador creado a partir de los datos de entrenamiento, que luego se aplican a los datos de prueba. Para la mayoría de los paquetes de aprendizaje automático, esto implica aplicar la función `predict()` a un objeto de modelo y un frame de datos de prueba, como: `predictions <- predict(model, test_data)`.

Hasta ahora, solo hemos examinado las predicciones de clasificación utilizando estos dos vectores de datos, pero la mayoría de los modelos pueden proporcionar otra pieza de información útil.

Aunque el clasificador hace una única predicción sobre cada ejemplo, puede estar más seguro de algunas decisiones que de otras. Por ejemplo, un clasificador puede estar 99 por ciento seguro de que un SMS con las palabras "free" y "ringtones" es spam, pero solo 51 por ciento seguro de que un SMS con la palabra "tonight" es spam. En ambos casos, el clasificador clasifica el mensaje como spam, pero está mucho más seguro de una decisión que de la otra.

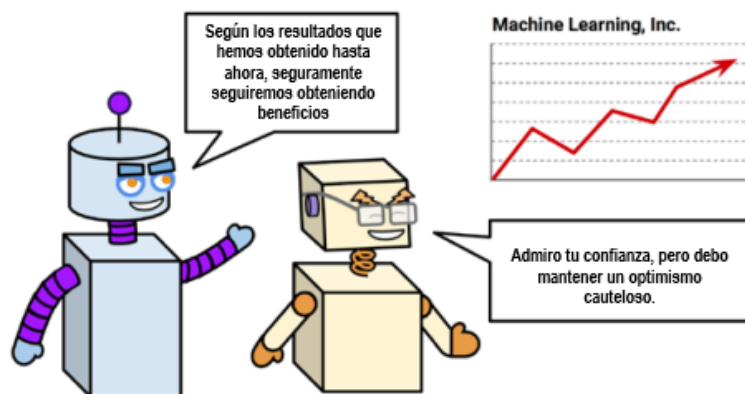


Figura 1: Los aprendices pueden diferir en su confianza de predicción incluso cuando se entrenan con los mismos datos.

El estudio de estas probabilidades de predicción internas proporciona datos útiles para evaluar el rendimiento de un modelo. Si dos modelos cometen la misma cantidad de errores, pero uno es más capaz de evaluar con mayor precisión su incertidumbre, entonces es un modelo más inteligente. Lo ideal es encontrar un aprendiz que tenga mucha confianza al hacer una predicción correcta, pero que sea tímido ante la duda. El equilibrio entre la confianza y la cautela es una parte clave de la evaluación del modelo.

La llamada a la función para obtener las probabilidades de predicción internas varía entre los paquetes de R. En general, para la mayoría de los clasificadores, la función `predict()` permite un parámetro adicional para especificar el tipo de predicción deseado. Para obtener una única clase predicha, como spam o ham, normalmente se establece el parámetro `type = "class"`. Para obtener la probabilidad de predicción, el parámetro `type` debe establecerse en uno de los siguientes: "prob", "posterior", "raw" o "probability", según el clasificador utilizado.

Todos los clasificadores presentados en este curso proporcionarán probabilidades de predicción. El parámetro `type` se incluye en el cuadro de sintaxis que presenta cada modelo.

Por ejemplo, para generar las probabilidades predichas para el clasificador C5.0 creado en el tema, Divide y vencerás: Clasificación mediante árboles de decisión y reglas, utiliza la función `predict()` con el tipo "prob" de la siguiente manera:

```
> predicted_prob <- predict(credit_model, credit_test, type = "prob")
```

Para generar las probabilidades predichas de Naive Bayes para el modelo de clasificación de spam de SMS desarrollado en el tema, Aprendizaje probabilístico: Clasificación mediante naïve Bayes, utiliza `predict()` con el tipo "raw" de la siguiente manera:

```
> sms_test_prob <- predict(sms_classifier, sms_test, type = "raw")
```

En la mayoría de los casos, la función `predict()` devuelve una probabilidad para cada categoría del resultado. Por ejemplo, en el caso de un modelo de dos resultados como el clasificador de SMS, las probabilidades predichas se pueden almacenar en una matriz o frame de datos, como se muestra aquí:

```
> head(sms_test_prob)
```

	ham	spam
[1,]	9.999995e-01	4.565938e-07
[2,]	9.999995e-01	4.540489e-07
[3,]	9.998418e-01	1.582360e-04
[4,]	9.999578e-01	4.223125e-05
[5,]	4.816137e-10	1.000000e+00
[6,]	9.997970e-01	2.030033e-04

Cada línea de este resultado muestra la probabilidad predicha del clasificador de spam y ham.

De acuerdo con las reglas de probabilidad, la suma de cada línea es uno porque estos son resultados mutuamente excluyentes y exhaustivos. Con estos datos, al construir el conjunto de datos de evaluación del modelo, es importante asegurarse de seleccionar solo la probabilidad para el nivel de clase de interés. Para mayor comodidad durante el proceso de evaluación, puede ser útil construir un frame de datos que recopile la clase predicha, la clase real y la probabilidad predicha del nivel de clase de interés.

Los pasos necesarios para construir el conjunto de datos de evaluación se han omitido por brevedad, pero se incluyen adjuntos. Para seguir los ejemplos aquí, descarga el archivo `sms_results.csv` y cárgalo en un frame de datos utilizando el comando

```
> sms_results<- read.csv("sms_results.csv", stringAsFactors = TRUE)
```

El frame de datos `sms_results` es simple. Contiene cuatro vectores de 1,390 valores.

Un vector contiene valores que indican el tipo real de mensaje SMS (spam o ham), un vector indica el tipo de mensaje predicho por el modelo Naïve Bayes y el tercer y cuarto vector indican la probabilidad de que el mensaje fuera spam o ham, respectivamente:

```
> head(sms_results)
```

	actual_type	predict_type	prob_spam	prob_ham
1	ham	ham	0.00000	1.00000
2	ham	ham	0.00000	1.00000
3	ham	ham	0.00016	0.99984
4	ham	ham	0.00004	0.99996
5	spam	spam	1.00000	0.00000
6	ham	ham	0.00020	0.99980

Para estos seis casos de prueba, los tipos de mensajes SMS predichos y reales coinciden; el modelo predijo su estado correctamente. Además, las probabilidades de predicción sugieren que el modelo estaba extremadamente seguro de estas predicciones porque todas están cerca de cero o uno.

¿Qué sucede cuando los valores predichos y reales están más lejos de cero y uno?

Usando la función `subset()`, podemos identificar algunos de estos registros. El siguiente resultado muestra casos de prueba donde el modelo estimó la probabilidad de spam entre 40 y 60 por ciento:

```
> head(subset(sms_results, prob_spam > 0.40 & prob_spam < 0.60))
```

```

      actual_type predict_type prob_spam prob_ham
377      spam      ham      0.47536 0.52464
717      ham      spam      0.56188 0.43812
1311     ham      spam      0.57917 0.42083

```

Según la propia estimación del modelo, estos fueron casos en los que una predicción correcta era prácticamente una moneda al aire. Sin embargo, las tres predicciones fueron erróneas, un resultado desafortunado. Veamos algunos casos más en los que el modelo estaba equivocado:

```
> head(subset(sms_results, actual_type != predict_type))
```

```

      actual_type predict_type prob_spam prob_ham
53      spam      ham      0.00071 0.99929
59      spam      ham      0.00156 0.99844
73      spam      ham      0.01708 0.98292
76      spam      ham      0.00851 0.99149
184     spam      ham      0.01243 0.98757
332     spam      ham      0.00003 0.99997

```

Estos casos ilustran el hecho importante de que un modelo puede ser extremadamente confiable y, sin embargo, puede estar extremadamente equivocado. Los seis casos de prueba eran mensajes de spam que el clasificador creía que tenían no menos del 98 por ciento de posibilidades de ser falsos.

A pesar de estos errores, ¿sigue siendo útil el modelo? Podemos responder a esta pregunta aplicando varias métricas de error a estos datos de evaluación. De hecho, muchas de estas métricas se basan en una herramienta que ya hemos utilizado ampliamente en temas anteriores.

Un análisis más detallado de las matrices de confusión

Una **matriz de confusión** es una tabla que clasifica las predicciones según sí coinciden o no con el valor real. Una de las dimensiones de la tabla indica las posibles categorías de valores predichos, mientras que la otra dimensión indica lo mismo para los valores reales. Aunque hasta ahora solo hemos visto matrices de confusión 2×2 , se puede crear una matriz para modelos que predicen cualquier número de valores de clase. La siguiente figura muestra la matriz de confusión habitual para un modelo binario de dos clases, así como la matriz de confusión 3×3 para un modelo de tres clases.

Cuando el valor predicho es el mismo que el valor real, se trata de una clasificación correcta. Las predicciones correctas se encuentran en la diagonal de la matriz de confusión (indicada por O). Las celdas de la matriz fuera de la diagonal (indicadas por X) indican los casos en los que el valor predicho difiere del valor real. Estas son predicciones incorrectas. Las

medidas de rendimiento de los modelos de clasificación se basan en los recuentos de predicciones que se encuentran dentro y fuera de la diagonal en estas tablas:













		Dos Clases				Tres Clases		
		Clase Predicha				Clase Predicha		
		A	B			A	B	C
Clase Real	A			Clase Real	A			
	B				B			
						C		

Figura 2: Las matrices de confusión cuentan los casos en los que la clase predicha coincide o no con el valor real.

Las medidas de rendimiento más comunes consideran la capacidad del modelo para discernir una clase frente a todas las demás. La clase de interés se conoce como clase **positiva**, mientras que todas las demás se conocen como **negativas**.

El uso de los términos positivo y negativo no pretende implicar ningún juicio de valor (es decir, bueno frente a malo), ni sugiere necesariamente que el resultado esté presente o ausente (como defecto de nacimiento frente a ninguno). La elección del resultado positivo puede incluso ser arbitraria, como en los casos en los que un modelo predice categorías como soleado frente a lluvioso, o perro frente a gato.

La relación entre las predicciones de clase positiva y clase negativa se puede representar como una matriz de confusión 2×2 que tabula si las predicciones caen en una de cuatro categorías:

- **Verdadero positivo** (TP, *True positive*): Clasificado correctamente como la clase de interés.
- **Verdadero negativo** (TN, *True negative*): Clasificado correctamente como no la clase de interés.
- **Falso positivo** (FP, *False positive*): Clasificado incorrectamente como la clase de interés.
- **Falso negativo** (FN, *False negative*): Clasificado incorrectamente como no la clase de interés.

Para el clasificador de spam, la clase positiva es spam, ya que este es el resultado que esperamos detectar. Luego podemos imaginar la matriz de confusión como se muestra en el siguiente diagrama:

		Se prevé que sea spam	
		no	si
En realidad es spam	no	<div>TN</div> Verdadero Negativo	<div>FP</div> Falso Positivo
	si	<div>FN</div> Falso Negativo	<div>TP</div> Verdadero Positivo

Figura 3: La distinción entre clases positivas y negativas agrega detalles a la matriz de confusión.

La matriz de confusión presentada de esta manera es la base de muchas de las medidas más importantes del rendimiento del modelo. En la siguiente sección, usaremos esta matriz para comprender mejor exactamente qué se entiende por precisión.

Uso de matrices de confusión para medir el rendimiento

Con la matriz de confusión 2×2 , podemos formalizar nuestra definición de **precisión de predicción** (a veces llamada **tasa de éxito**) como:

$$\text{accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

En esta fórmula, los términos TP, TN, FP y FN se refieren a la cantidad de veces que las predicciones del modelo cayeron en cada una de estas categorías. Por lo tanto, la precisión (*accuracy*) es una proporción que representa la cantidad de verdaderos positivos y verdaderos negativos dividida por la cantidad total de predicciones.

La **tasa de error**, o la proporción de ejemplos clasificados incorrectamente, se especifica como:

$$\text{error rate} = \frac{FP+FN}{TP+TN+FP+FN} = 1 - \text{accuracy}$$

Observa que la tasa de error se puede calcular como uno menos la precisión. Intuitivamente, esto tiene sentido; un modelo que es correcto el 95 por ciento del tiempo es incorrecto el cinco por ciento del tiempo.

Una forma sencilla de tabular las predicciones de un clasificador en una matriz de confusión es usar la función `table()` de R. El comando para crear una matriz de confusión para los datos de SMS se muestra a continuación. Los recuentos de esta tabla se pueden utilizar para calcular la precisión y otras estadísticas:

```
> table(sms_results$actual_type, sms_results$predict_type)
```

```
      ham spam
ham  1203   4
spam   31 152
```

Si deseas crear una matriz de confusión con una salida más informativa, la función `CrossTable()` del paquete `gmodels` ofrece una solución personalizable.

Si recuerdas, utilizamos esta función por primera vez en el documento, Gestión y comprensión de los datos. Si no instalaste el paquete en ese momento, deberás hacerlo mediante el comando `install.packages("gmodels")`.

De forma predeterminada, la salida de `CrossTable()` incluye proporciones en cada celda que indican el recuento de celdas como un porcentaje de los recuentos totales de filas, columnas y generales de la tabla. La salida también incluye los totales de filas y columnas. Como se muestra en el siguiente código, la sintaxis es similar a la de la función `table()`:

```
> library(gmodels)
```

```
> CrossTable(sms_results$actual_type, sms_results$predict_type)
```

El resultado es una matriz de confusión con una gran cantidad de detalles adicionales:

```
Cell Contents
|-----|
|              N              |
| Chi-square contribution |
|      N / Row Total      |
|      N / Col Total      |
|      N / Table Total    |
|-----|

Total Observations in Table:  1390

sms_results$actual_type | sms_results$predict_type
                        |      ham      |      spam      | Row Total |
-----|-----|-----|-----|
                        |      ham      |      spam      |      ham      |
ham |      1203      |         4      |      1207      |
      16.128      |    127.580     |
      0.997       |      0.003     |      0.868     |
      0.975       |      0.026     |
      0.865       |      0.003     |
-----|-----|-----|-----|
                        |      ham      |      spam      |      spam      |
spam |         31      |     152        |      183       |
      106.377     |    841.470     |
      0.169       |      0.831     |      0.132     |
      0.025       |      0.974     |
      0.022       |      0.109     |
-----|-----|-----|-----|
Column Total |      1234      |      156       |      1390      |
              |      0.888     |      0.112     |
-----|-----|-----|-----|
```

Hemos utilizado `CrossTable()` en varios temass anteriores, por lo que a esta altura ya deberías estar familiarizado con la salida. Si alguna vez olvidas cómo interpretar el resultado, simplemente consulta la llave (etiquetada Contenido de la celda), que proporciona la definición de cada número en las celdas de la tabla.

Podemos utilizar la matriz de confusión para obtener la precisión y la tasa de error. Dado que la precisión es $(TP + TN) / (TP + TN + FP + FN)$, podemos calcularla de la siguiente manera:

```
> (152 + 1203) / (152 + 1203 + 4 + 31)
[1] 0.9748201
```

También podemos calcular la tasa de error $(FP + FN) / (TP + TN + FP + FN)$ como:

```
> (4 + 31) / (152 + 1203 + 4 + 31)
[1] 0.02517986
```

Esto es lo mismo que uno menos la precisión:

```
> 1 - 0.9748201
[1] 0.0251799
```

Aunque estos cálculos pueden parecer simples, es importante practicar el pensamiento sobre cómo se relacionan entre sí los componentes de la matriz de confusión. En la siguiente sección, verás cómo estas mismas piezas se pueden combinar de diferentes maneras para crear una variedad de medidas de rendimiento adicionales.

Más allá de la precisión - otras medidas de rendimiento

Se han desarrollado y utilizado innumerables medidas de rendimiento para fines específicos en disciplinas tan diversas como la medicina, la recuperación de información, el marketing y la teoría de detección de señales, entre otras. Cubrirlas todas podría llenar cientos de páginas, lo que hace que una descripción exhaustiva aquí no sea factible. En cambio, consideraremos solo algunas de las medidas más útiles y más citadas en la literatura de aprendizaje automático.

El paquete de entrenamiento de clasificación y regresión `caret` de Max Kuhn incluye funciones para calcular muchas de estas medidas de rendimiento. Este paquete proporciona una gran cantidad de herramientas para preparar, entrenar, evaluar y visualizar modelos y datos de aprendizaje automático. Además de su uso aquí, también emplearemos `caret` ampliamente en el siguiente documento, Mejora del rendimiento del modelo. Antes de continuar, deberás instalar el paquete mediante el comando `install.packages("caret")`.

El paquete caret agrega otra función para crear una matriz de confusión. Como se muestra en el siguiente comando, la sintaxis es similar a `table()`, pero con una pequeña diferencia. Debido a que caret calcula medidas del rendimiento del modelo que reflejan la capacidad de clasificar la clase positiva, se debe especificar un parámetro positivo. En este caso, dado que el clasificador SMS está destinado a detectar spam, estableceremos `positive = "spam"` de la siguiente manera:

```
> library(caret)
> confusionMatrix(sms_results$predict_type,
+ sms_results$actual_type, positive = "spam")
```

Esto da como resultado el siguiente resultado:

```
Confusion Matrix and Statistics

      Reference
Prediction ham spam
ham      1203    31
spam       4    152

      Accuracy : 0.9748
      95% CI   : (0.9652, 0.9824)
No Information Rate : 0.8683
P-Value [Acc > NIR] : < 2.2e-16

      Kappa : 0.8825

McNemar's Test P-Value : 1.109e-05

      Sensitivity : 0.8306
      Specificity : 0.9967
      Pos Pred Value : 0.9744
      Neg Pred Value : 0.9749
      Prevalence : 0.1317
      Detection Rate : 0.1094
      Detection Prevalence : 0.1122
      Balanced Accuracy : 0.9136

      'Positive' Class : spam
```

En la parte superior del resultado hay una matriz de confusión muy similar a la producida por la función `table()`, pero transpuesta. El resultado también incluye un conjunto de medidas de rendimiento. Algunas de ellas, como la precisión, son conocidas, mientras que muchas otras son nuevas. Echemos un vistazo a algunas de las métricas más importantes.

La estadística kappa

La estadística kappa (denominada Kappa en el resultado anterior) ajusta la precisión teniendo en cuenta la posibilidad de una predicción correcta solo por casualidad. Esto es especialmente importante para conjuntos de datos con un desequilibrio de clases grave porque un clasificador puede obtener una alta precisión simplemente adivinando siempre la

clase más frecuente. La estadística kappa solo recompensará al clasificador si acierta con más frecuencia que esta estrategia simplista.

Hay más de una forma de definir la estadística kappa. El método más común, descrito aquí, utiliza el coeficiente kappa de Cohen, como se describe en el artículo: A coefficient of agreement for nominal scales, Cohen, J, Education and Psychological Measurement, 1960, Vol. 20, pp. 37-46.

Los valores kappa suelen oscilar entre 0 y un máximo de 1, los valores más altos reflejan una mayor concordancia entre las predicciones del modelo y los valores reales. Es posible observar valores menores que 0 si las predicciones apuntan constantemente en la dirección incorrecta, es decir, si las predicciones no coinciden con los valores reales o son erróneas con más frecuencia de lo que cabría esperar si se hiciera una suposición aleatoria.

Esto rara vez ocurre en los modelos de aprendizaje automático y, por lo general, refleja un problema de codificación, que se puede solucionar simplemente invirtiendo las predicciones.

Según cómo se vaya a utilizar un modelo, la interpretación de la estadística kappa puede variar. Una interpretación común se muestra de la siguiente manera:

- Concordancia deficiente = menor a 0.20
- Concordancia aceptable = 0.20 a 0.40
- Concordancia moderada = 0.40 a 0.60
- Concordancia buena = 0.60 a 0.80
- Concordancia muy buena = 0.80 a 1.00

Es importante tener en cuenta que estas categorías son subjetivas. Si bien una “concordancia buena” puede ser más que adecuada para predecir el sabor de helado favorito de alguien, una “concordancia muy buena” puede no ser suficiente si su objetivo es identificar defectos de nacimiento.

Para obtener más información sobre la escala anterior, consulta: The measurement of observer agreement for categorical data, Landis, JR, Koch, GG. Biometrics, 1997, vol. 33, págs. 159-174.

La siguiente es la fórmula para calcular la estadística kappa. En esta fórmula, $\Pr(a)$ se refiere a la proporción de acuerdo real y $\Pr(e)$ se refiere al acuerdo esperado entre el clasificador y los valores verdaderos, bajo el supuesto de que fueron elegidos al azar:

$$\kappa = \frac{\Pr(a) - \Pr(e)}{1 - \Pr(e)}$$

Estas proporciones son fáciles de obtener a partir de una matriz de confusión una vez que se sabe dónde buscar. Consideremos la matriz de confusión para el modelo de clasificación de SMS creado con la función `CrossTable()`, que se repite aquí para mayor comodidad:

sms_results\$actual_type	sms_results\$predict_type		Row Total
	ham	spam	
ham	1203	4	1207
	16.128	127.580	
	0.997	0.003	0.868
	0.975	0.026	
	0.865	0.003	
spam	31	152	183
	106.377	841.470	
	0.169	0.831	0.132
	0.025	0.974	
	0.022	0.109	
Column Total	1234	156	1390
	0.888	0.112	

Recuerda que el valor inferior de cada celda indica la proporción de todas las instancias que caen en esa celda. Por lo tanto, para calcular la concordancia observada $Pr(a)$, simplemente sumamos la proporción de todas las instancias en las que el tipo predicho y el tipo real de SMS coinciden. Por lo tanto, podemos calcular $Pr(a)$ como:

```
> pr_a <- 0.865 + 0.109
> pr_a
[1] 0.974
```

Para este clasificador, los valores observados y reales coinciden el 97.4 por ciento de las veces; notarás que esto es lo mismo que la precisión (*accuracy*). La estadística kappa ajusta la precisión en relación con la concordancia esperada, $Pr(e)$, que es la probabilidad de que el azar por sí solo haga que los valores predichos y reales coincidan, bajo el supuesto de que ambos se seleccionan aleatoriamente de acuerdo con las proporciones observadas.

Para encontrar estas proporciones observadas, podemos usar las reglas de probabilidad que aprendimos en el documento, Aprendizaje probabilístico: Clasificación mediante naïve Bayes. Suponiendo que dos eventos son independientes (es decir, uno no afecta al otro), las reglas de probabilidad indican que la probabilidad de que ambos ocurran es igual al producto de las probabilidades de que cada uno ocurra. Por ejemplo, sabemos que la probabilidad de que ambos elijan ham es:

$Pr(\text{actual_type is ham}) * Pr(\text{predicted_type is ham})$

Y la probabilidad de que ambos elijan spam es:

$\Pr(\text{actual_type is spam}) * \Pr(\text{predicted_type is spam})$

La probabilidad de que el tipo predicho o real sea spam o ham se puede obtener a partir de los totales de filas o columnas. Por ejemplo, $\Pr(\text{actual_type is ham}) = 0.868$ y $\Pr(\text{predicted_type is ham}) = 0.888$.

$\Pr(e)$ se puede calcular como la suma de las probabilidades de que los valores predichos y reales coincidan en que el mensaje es spam o ham. Recordemos que, en el caso de eventos mutuamente excluyentes (eventos que no pueden ocurrir simultáneamente), la probabilidad de que ocurra uno de ellos es igual a la suma de sus probabilidades. Por lo tanto, para obtener el $\Pr(e)$ final, simplemente sumamos ambos productos, de la siguiente manera:

```
> pr_e <- 0.868 * 0.888 + 0.132 * 0.112
> pr_e
[1] 0.785568
```

Como $\Pr(e)$ es 0.786, por pura casualidad esperaríamos que los valores observados y reales coincidieran aproximadamente el 78.6 por ciento de las veces.

Esto significa que ahora tenemos toda la información necesaria para completar la fórmula kappa. Al introducir los valores $\Pr(a)$ y $\Pr(e)$ en la fórmula kappa, obtenemos:

```
> k <- (pr_a - pr_e) / (1 - pr_e)
> k
[1] 0.8787494
```

El kappa es aproximadamente 0.88, lo que coincide con el resultado previo de `confusionMatrix()` de `caret` (la pequeña diferencia se debe al redondeo). Utilizando la interpretación sugerida, observamos que hay una muy buena concordancia entre las predicciones del clasificador y los valores reales.

Hay un par de funciones R para calcular kappa automáticamente. La función `Kappa()` (asegúrate de tener en cuenta la "K" mayúscula) del paquete `Visualizing Categorical Data (vcd)` utiliza una matriz de confusión de valores previstos y reales. Después de instalar el paquete escribiendo `install.packages("vcd")`, se pueden utilizar los siguientes comandos para obtener kappa:

```
> install.packages("vcd")
> library(vcd)
> Kappa(table(sms_results$actual_type, sms_results$predict_type))
```

	value	ASE	z	Pr(> z)
Unweighted	0.8825	0.01949	45.27	0
Weighted	0.8825	0.01949	45.27	0

Nos interesa el kappa no ponderado. El valor de 0.88 coincide con lo que calculamos a mano.

El kappa ponderado se utiliza cuando hay distintos grados de acuerdo. Por ejemplo, si se utiliza una escala de frío, fresco, templado y caliente, un valor de cálido concuerda más con caliente que con el valor de frío. En el caso de un evento con dos resultados, como spam y ham, las estadísticas kappa ponderadas y no ponderadas serán idénticas.

La función `kappa2()` del paquete `Interrater Reliability (irr)` se puede utilizar para calcular kappa a partir de vectores de valores previstos y reales almacenados en un frame de datos. Después de instalar el paquete mediante `install.packages("irr")`, se pueden utilizar los siguientes comandos para obtener el kappa:

```
> install.packages("irr")
> library(irr)
> kappa2(sms_results[1:2])

Cohen's Kappa for 2 Raters (Weights: unweighted)

Subjects = 1390
Raters = 2
Kappa = 0.883

z = 33
p-value = 0
```

Las funciones `Kappa()` y `kappa2()` informan la misma estadística kappa, así que utiliza la opción con la que se sienta más cómodo.

Ten cuidado de no utilizar la función `kappa()` incorporada. ¡No tiene ninguna relación con la estadística kappa informada anteriormente!

El coeficiente de correlación de Matthews

Aunque la precisión y el kappa han sido medidas populares de rendimiento durante muchos años, una tercera opción se ha convertido rápidamente en un estándar de facto en el campo del aprendizaje automático. Al igual que las dos métricas anteriores, el coeficiente de correlación de Matthews (MCC, **Matthews correlation coefficient**) es una estadística única destinada a reflejar el rendimiento general de un modelo de clasificación. Además, el MCC es como kappa en el sentido de que es útil incluso en el caso de que el conjunto de datos esté gravemente desequilibrado, el tipo de situaciones en las que la medida de precisión tradicional puede ser muy engañosa.

El MCC ha ganado popularidad debido a su facilidad de interpretación, así como a un creciente conjunto de evidencia que sugiere que funciona mejor en una variedad más amplia de circunstancias que kappa. Investigaciones empíricas recientes han indicado que el MCC puede ser la mejor métrica única para describir el rendimiento en el mundo real de un modelo de clasificación binaria. Otros estudios han identificado circunstancias potenciales en las que la estadística kappa proporciona una descripción engañosa o incorrecta del rendimiento del modelo. En estos casos, cuando el MCC y kappa no coinciden, la métrica MCC tiende a proporcionar una evaluación más razonable de las verdaderas capacidades del modelo.

Para obtener más información sobre las ventajas relativas del coeficiente de correlación de Matthews frente a kappa, consulta: The Matthews correlation equilibrium (MCC) is more informative than Cohen's kappa and brier score in binary classification assessment, Chicco D, Warrens MJ, Jurman G, IEEE Access, 2021, vol. 9, págs. 78368-78381. Alternativamente, consulta: Why Cohen's Kappa should be avoided as performance measure in classification, Delgado R, Tibau XA, PLoS One, 2019, vol. 14(9).

Los valores del MCC se interpretan en la misma escala que el coeficiente de correlación de Pearson, que se presentó en el documento, Pronóstico de datos numéricos: Métodos de regresión. Este valor varía de -1 a +1, lo que indica predicciones perfectamente inexactas y perfectamente precisas, respectivamente. Un valor de 0 indica un modelo que no funciona mejor que una suposición aleatoria. Como la mayoría de las puntuaciones de MCC se encuentran en algún lugar en el rango de valores entre 0 y 1, hay cierta subjetividad involucrada en saber qué es una puntuación “buena”. Al igual que la escala utilizada para las correlaciones de Pearson, una posible interpretación es la siguiente:

- Perfectamente incorrecto = -1.0
- Fuertemente incorrecto = -0.5 a -1.0
- Moderadamente incorrecto = -0.3 a -0.5
- Débilmente incorrecto = -0.1 a 0.3
- Aleatoriamente correcto = -0.1 a 0.1
- Débilmente correcto = 0.1 a 0.3
- Moderadamente correcto = 0.3 a 0.5
- Fuertemente correcto = 0.5 a 1.0
- Perfectamente correcto = 1.0

Observa que los modelos con peor desempeño se encuentran en la mitad de la escala. En otras palabras, un modelo en el lado negativo de la escala (de perfectamente a débilmente incorrecto) todavía funciona mejor que un modelo que predice al azar. Por ejemplo, aunque la precisión de un modelo fuertemente incorrecto sea baja, las predicciones pueden simplemente revertirse para obtener el resultado correcto.

Al igual que con todas las escalas de este tipo, estas deben usarse solo como pautas aproximadas. Además, el beneficio clave de una métrica como el MCC no es comprender el desempeño de un modelo de manera aislada, sino más bien, facilitar las comparaciones de desempeño entre varios modelos.

El MCC se puede calcular a partir de la matriz de confusión para un clasificador binario como se muestra en la siguiente fórmula:

$$\text{MCC} = \frac{\text{TP} \times \text{TN} - \text{FP} \times \text{FN}}{\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})}}$$

Usando la matriz de confusión para el modelo de clasificación de spam SMS, obtenemos los siguientes valores:

- TN = 1203
- FP = 4
- FN = 31
- TP = 152

El MCC se puede calcular manualmente en R de la siguiente manera:

```
> (152 * 1203 - 4 * 31) /
+ sqrt((152 + 4) * (152 + 31) * (1203 + 4) * (1203 + 31))
[1] 0.8861669
```

El paquete mltools de Ben Gorman proporciona una función `mcc()` que puede realizar el cálculo de MCC utilizando vectores de valores predichos y reales. Después de instalar el paquete, el siguiente código R produce el mismo resultado que el cálculo realizado a mano:

```
> install.packages("mltools")
> library(mltools)
> mcc(sms_results$actual_type, sms_results$predict_type)
[1] 0.8861669
```

Alternativamente, para un clasificador binario donde la clase positiva se codifica como 1 y la clase negativa se codifica como 0, el MCC es idéntico a la correlación de Pearson entre los valores predichos y reales. Podemos demostrar esto utilizando la función `cor()` en R, después de utilizar `ifelse()` para convertir los valores categóricos ("spam" o "ham") en valores binarios (1 o 0) de la siguiente manera:

```
> cor(ifelse(sms_results$actual_type == "spam", 1, 0),
+ ifelse(sms_results$predict_type == "spam", 1, 0))
[1] 0.8861669
```

El hecho de que una métrica de rendimiento de clasificación tan obvia estuviera oculta a simple vista, como una simple adaptación de la correlación de Pearson introducida a fines del siglo XIX, hace que sea bastante notable que el MCC solo se haya vuelto popular en las últimas décadas. El bioquímico Brian W. Matthews es responsable de popularizar esta métrica en 1975 para su uso en problemas de clasificación de dos resultados y, por lo tanto, recibe el crédito por nombrar esta aplicación específica. Sin embargo, parece bastante probable que la métrica ya se usara ampliamente, incluso si no había ganado mucha atención hasta mucho después.

Hoy en día, se utiliza en la industria, la investigación académica e incluso como un punto de referencia para las competencias de aprendizaje automático. Puede que no haya una única métrica que capture mejor el rendimiento general de un modelo de clasificación binaria. Sin embargo, como verás pronto, se puede obtener una comprensión más profunda del rendimiento del modelo utilizando combinaciones de métricas.

Aunque el MCC se define aquí para la clasificación binaria, no está claro si es la mejor métrica para los resultados de múltiples clases. Para una discusión de esta y otras alternativas, consulta: A comparison of MCC and CEN error measures in multi-class prediction, Jurman G, Riccadonna S, Furlanello C, 2012, PLOS One 7(8).

Sensibilidad y especificidad

Encontrar un clasificador útil a menudo implica un equilibrio entre predicciones demasiado conservadoras y demasiado agresivas. Por ejemplo, un filtro de correo electrónico podría garantizar la eliminación de todos los mensajes de spam eliminando agresivamente casi todos los mensajes de radioaficionados. Por otro lado, para garantizar que ningún mensaje de radioaficionado se filtre inadvertidamente, podríamos tener que permitir que una cantidad inaceptable de spam pase a través del filtro. Un par de medidas de rendimiento capturan esta disyuntiva: sensibilidad y especificidad.

La **sensibilidad** de un modelo (también llamada **tasa de verdaderos positivos**) mide la proporción de ejemplos positivos que se clasificaron correctamente. Por lo tanto, como se muestra en la siguiente fórmula, se calcula como el número de verdaderos positivos dividido por el número total de positivos, tanto los clasificados correctamente (los verdaderos positivos) como los clasificados incorrectamente (los falsos negativos):

$$\text{sensitivity} = \frac{TP}{TP + FN}$$

La **especificidad** de un modelo (también llamada **tasa de verdaderos negativos**) mide la proporción de ejemplos negativos que se clasificaron correctamente. Al igual que con la

sensibilidad, esto se calcula como el número de verdaderos negativos dividido por el número total de negativos (los verdaderos negativos más los falsos positivos).

$$\text{specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

Dada la matriz de confusión para el clasificador de SMS, podemos calcular fácilmente estas medidas a mano. Suponiendo que el spam es la clase positiva, podemos confirmar que los números en la salida de `confusionMatrix()` son correctos. Por ejemplo, el cálculo para la sensibilidad es:

```
> sens <- 152 / (152 + 31)
> sens
[1] 0.8306011
```

De manera similar, para la especificidad podemos calcular:

```
> spec <- 1203 / (1203 + 4)
> spec
[1] 0.996686
```

El paquete `caret` proporciona funciones para calcular la sensibilidad y la especificidad directamente a partir de vectores de valores predichos y reales. Ten cuidado de especificar el parámetro positivo o negativo de manera apropiada, como se muestra en las siguientes líneas:

```
> library(caret)
> sensitivity(sms_results$predict_type, sms_results$actual_type,
+   positive = "spam")
[1] 0.8306011
> specificity(sms_results$predict_type, sms_results$actual_type,
+   negative = "ham")
[1] 0.996686
```

La sensibilidad y la especificidad varían de cero a uno, siendo más deseables los valores cercanos a uno. Por supuesto, es importante encontrar un equilibrio apropiado entre los dos, una tarea que a menudo es bastante específica del contexto.

Por ejemplo, en este caso, la sensibilidad de 0.831 implica que el 83.1 por ciento de los mensajes de spam se clasificaron correctamente. De manera similar, la especificidad de 0.997 implica que el 99.7 por ciento de los mensajes que no eran spam se clasificaron correctamente o, alternativamente, el 0.3 por ciento de los mensajes válidos se rechazaron como spam. La

idea de rechazar el 0.3 por ciento de los mensajes SMS válidos puede ser inaceptable, o puede ser una compensación razonable dada la reducción del spam.

La sensibilidad y la especificidad proporcionan herramientas para pensar en tales compensaciones. Por lo general, se realizan cambios en el modelo y se prueban diferentes modelos hasta encontrar uno que cumpla con un umbral de sensibilidad y especificidad deseado. Las visualizaciones, como las que se analizan más adelante en este documento, también pueden ayudar a comprender el equilibrio entre la sensibilidad y la especificidad.

Precisión y recuperación

Estrechamente relacionadas con la sensibilidad y la especificidad hay otras dos medidas de rendimiento relacionadas con los compromisos realizados en la clasificación: la precisión (*precision*) y la recuperación (*recall*). Utilizadas principalmente en el contexto de la recuperación de la información, estas estadísticas tienen como objetivo indicar cuán interesantes y relevantes son los resultados de un modelo, o si las predicciones se diluyen por ruido sin sentido.

La **precisión** (también conocida como **valor predictivo positivo**) se define como la proporción de ejemplos positivos que son verdaderamente positivos; en otras palabras, cuando un modelo predice la clase positiva, ¿con qué frecuencia es correcto? Un modelo preciso solo predecirá la clase positiva en casos con muchas probabilidades de ser positivos. Será muy confiable.

$$\text{precision} = \frac{TP}{TP + FP}$$

Considera lo que sucedería si el modelo fuera muy impreciso. Con el tiempo, los resultados serían menos confiables. En el contexto de la recuperación de la información, esto sería similar a un motor de búsqueda como Google que devuelve resultados no relacionados. Con el tiempo, los usuarios cambiarían a un competidor como Bing. En el caso del filtro de spam de SMS, alta precisión significa que el modelo es capaz de apuntar cuidadosamente solo al spam mientras ignora al ham.

Por otra parte, la **recuperación** es una medida de cuán completos son los resultados. Como se muestra en la siguiente fórmula, esto se define como el número de verdaderos positivos sobre el número total de positivos. Es posible que ya hayas reconocido que esto es lo mismo que la sensibilidad; sin embargo, la interpretación difiere ligeramente.

$$\text{recall} = \frac{TP}{TP + FN}$$

Un modelo con alta recuperación captura una gran parte de los ejemplos positivos, lo que significa que tiene una amplia amplitud. Por ejemplo, un motor de búsqueda con alta recuperación devuelve una gran cantidad de documentos pertinentes a la consulta de búsqueda. De manera similar, el filtro de spam SMS tiene alta recuperación si la mayoría de los mensajes de spam se identifican correctamente.

Podemos calcular la precisión y la recuperación a partir de la matriz de confusión. Nuevamente, suponiendo que el spam es la clase positiva, la precisión es:

```
> prec <- 152 / (152 + 4)
> prec
[1] 0.974359
```

Y la recuperación es:

```
> rec <- 152 / (152 + 31)
> rec
[1] 0.8306011
```

El paquete caret se puede utilizar para calcular cualquiera de estas medidas a partir de vectores de clases previstas y reales. La precisión utiliza la función posPredValue():

```
> library(caret)
> posPredValue(sms_results$predict_type, sms_results$actual_type,
+   positive = "spam")
[1] 0.974359
```

La recuperación utiliza la función sensitive() que utilizamos anteriormente:

```
> sensitivity(sms_results$predict_type, sms_results$actual_type,
+   positive = "spam")
[1] 0.8306011
```

De manera similar a la disyuntiva inherente entre sensibilidad y especificidad, para la mayoría de los problemas del mundo real, es difícil construir un modelo con alta precisión y alta recuperación. Es fácil ser preciso si se apunta solo a los frutos más fáciles de alcanzar - los ejemplos fáciles de clasificar.

De manera similar, es fácil que un modelo tenga alta recuperación si se lanza una red muy amplia, lo que significa que el modelo es demasiado agresivo en la identificación de los casos positivos. Por el contrario, tener alta precisión y recuperación al mismo tiempo es muy

desafiante. Por lo tanto, es importante probar una variedad de modelos para encontrar la combinación de precisión y recuperación que satisfaga las necesidades de tu proyecto.

La medida F

Una medida del rendimiento del modelo que combina precisión y recuperación en un único número se conoce como medida F (también llamada a veces **puntuación F₁** (*F₁ score*) o **puntuación F** (*F-score*)). La medida F combina precisión y recuperación utilizando la **media armónica**, un tipo de promedio que se utiliza para las tasas de cambio. Se utiliza la media armónica en lugar de la media aritmética más común, ya que tanto la precisión como la recuperación se expresan como proporciones entre cero y uno, que se pueden interpretar como tasas. La siguiente es la fórmula para la medida F:

$$F\text{-measure} = \frac{2 \times \text{precision} \times \text{recall}}{\text{recall} + \text{precision}} = \frac{2 \times TP}{2 \times TP + FP + FN}$$

Para calcular la medida F, utilice los valores de precisión y recuperación calculados anteriormente:

```
> f <- (2 * prec * rec) / (prec + rec)
> f
[1] 0.8967552
```

Esto resulta exactamente igual que utilizar los recuentos de la matriz de confusión:

```
> f <- (2 * 152) / (2 * 152 + 4 + 31)
> f
[1] 0.8967552
```

Dado que la medida F describe el rendimiento del modelo en un único número, proporciona una forma conveniente de comparar varios modelos uno al lado del otro. Sin embargo, esto supone que se debe asignar el mismo peso a la precisión y la recuperación, una suposición que no siempre es válida.

Es posible calcular los puntajes F utilizando diferentes ponderaciones para la precisión y la recuperación, pero elegir los pesos puede ser complicado en el mejor de los casos y arbitrario en el peor.

Una mejor práctica es utilizar medidas como el puntaje F (F-score) en combinación con métodos que consideren las fortalezas y debilidades de un modelo de manera más global, como los que se describen en la siguiente sección.

Visualización de compensaciones de rendimiento con curvas ROC

Las visualizaciones son útiles para comprender el rendimiento de los algoritmos de aprendizaje automático con mayor detalle. Mientras que las estadísticas como la sensibilidad y la especificidad, o la precisión y la recuperación, intentan reducir el rendimiento del modelo a un solo número, las visualizaciones representan cómo se desempeña un aprendiz en una amplia gama de condiciones.

Debido a que los algoritmos de aprendizaje tienen diferentes sesgos, es posible que dos modelos con precisión similar puedan tener diferencias drásticas en la forma en que logran su precisión. Algunos modelos pueden tener dificultades con ciertas predicciones que otros hacen con facilidad, mientras que superan rápidamente los casos que otros no pueden acertar.

Las visualizaciones proporcionan un método para comprender estas compensaciones al comparar a los aprendices uno al lado del otro en un solo gráfico.

La curva ROC (característica operativa del receptor, **receiver operating characteristic**) se utiliza habitualmente para examinar la compensación entre la detección de verdaderos positivos y la evitación de falsos positivos. Como puedes sospechar por el nombre, las curvas ROC fueron desarrolladas por ingenieros en el campo de las comunicaciones.

En la época de la Segunda Guerra Mundial, los operadores de radar y radio utilizaban curvas ROC para medir la capacidad de un receptor de discriminar entre señales verdaderas y falsas alarmas. La misma técnica es útil hoy en día para visualizar la eficacia de los modelos de aprendizaje automático.

Para obtener más información sobre las curvas ROC, consulta: An introduction to ROC analysis, Fawcett T, Pattern Recognition Letters, 2006, vol. 27, pp. 861–874.

Las características de un diagrama ROC típico se representan en el siguiente gráfico (ver la figura 4).

La figura se dibuja utilizando la proporción de verdaderos positivos en el eje vertical y la proporción de falsos positivos en el eje horizontal. Debido a que estos valores son equivalentes a la sensibilidad y $(1 - \text{especificidad})$ respectivamente, el diagrama también se conoce como gráfico de sensibilidad/especificidad.

Los puntos que componen las curvas ROC indican la tasa de verdaderos positivos en distintos umbrales de falsos positivos. Para ilustrar este concepto, en el gráfico se contrastan tres clasificadores hipotéticos.

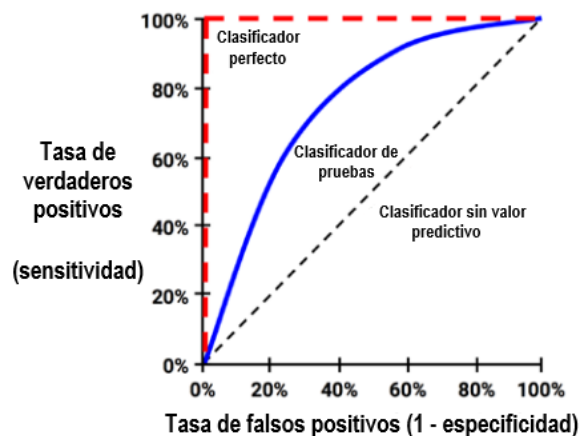


Figura 4: La curva ROC representa las formas de los clasificadores en relación con los clasificadores perfectos e inútiles.

En primer lugar, el *clasificador perfecto* tiene una curva que pasa por el punto con una tasa de verdaderos positivos del 100 por ciento y una tasa de falsos positivos del 0 por ciento. Es capaz de identificar correctamente todos los verdaderos positivos antes de clasificar incorrectamente cualquier resultado negativo. A continuación, la línea diagonal que va desde la esquina inferior izquierda hasta la esquina superior derecha del diagrama representa un *clasificador sin valor predictivo*.

Este tipo de clasificador detecta verdaderos positivos y falsos positivos exactamente con la misma tasa, lo que implica que el clasificador no puede discriminar entre los dos. Esta es la línea de base por la que se pueden juzgar otros clasificadores. Las curvas ROC que caen cerca de esta línea indican modelos que no son muy útiles. Por último, la mayoría de los clasificadores del mundo real son como el clasificador de prueba, en el sentido de que se encuentran en algún lugar en la zona entre perfectos e inútiles.

La mejor manera de entender cómo se construye la curva ROC es crearla a mano. Los valores de la tabla que se muestran en la figura 5 indican las predicciones de un modelo hipotético de spam aplicado a un conjunto de prueba que contiene 20 ejemplos, de los cuales seis son de la clase positiva (spam) y 14 son de la clase negativa (ham).

Para crear las curvas, las predicciones del clasificador se ordenan por la probabilidad estimada del modelo de la clase positiva, en orden descendente, con los valores más grandes primero, como se muestra en la tabla.

Luego, comenzando en el origen del gráfico, el impacto de cada predicción en la tasa de positivos verdaderos y la tasa de positivos falsos da como resultado una curva que traza verticalmente para cada ejemplo positivo y horizontalmente para cada ejemplo negativo. Este

proceso se puede realizar a mano en una hoja de papel cuadriculado, como se muestra en la figura 6:

Estimated Spam Probability	Actual Message Type
0.95	spam
0.90	spam
0.75	spam
0.70	spam
0.60	ham
0.55	spam
0.51	ham
0.49	spam
0.38	ham
0.35	ham
0.30	ham
0.25	ham
0.21	ham
0.20	ham
0.19	ham
0.19	ham
0.18	ham
0.15	ham
0.11	ham
0.10	ham

Figura 5: Para construir la curva ROC, los valores de probabilidad estimados para la clase positiva se ordenan en orden descendente y luego se comparan con el valor de la clase real.

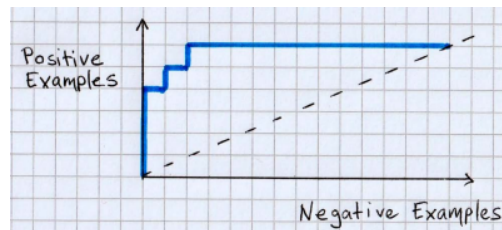


Figura 6: La curva ROC se puede crear a mano en papel cuadriculado, trazando el número de ejemplos positivos frente al número de ejemplos negativos.

Obsérvese que la curva ROC no está completa en este punto, porque los ejes están sesgados debido a la presencia de más del doble de ejemplos negativos que de ejemplos positivos en el conjunto de prueba. Una solución sencilla para esto es escalar el gráfico proporcionalmente de modo que los dos ejes sean equivalentes en tamaño, como se muestra en la figura 7.

Si imaginamos que los ejes x e y ahora van de 0 a 1, podemos interpretar cada eje como un porcentaje. El eje y es el número de positivos y originalmente iba de 0 a 6; después de reducirlo a una escala de 0 a 1, cada incremento se convierte en $1/6$. En esta escala, podemos pensar en la coordenada vertical de la curva ROC como el número de verdaderos positivos dividido por el número total de positivos, que es la tasa de verdaderos positivos o sensibilidad. De manera similar, el eje x mide el número de negativos; al dividir por el número

total de negativos (14 en este ejemplo), obtenemos la tasa de verdaderos negativos o especificidad.

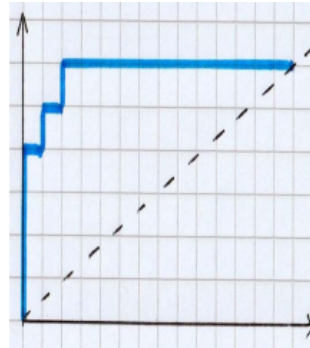


Figura 7: Escalar los ejes del gráfico crea una comparación proporcional, independientemente del balance inicial de ejemplos positivos y negativos.

La tabla de la figura 8 muestra estos cálculos para los 20 ejemplos del conjunto de prueba hipotético:

Example Spam Model						
Predicted Prob. of Spam	Actual Type	True Positives	True Negatives	TP Rate (Sensitivity)	TN Rate (Specificity)	FP Rate (1 - Specificity)
0.95	spam	1	14	16.7%	100.0%	0.0%
0.90	spam	2	14	33.3%	100.0%	0.0%
0.75	spam	3	14	50.0%	100.0%	0.0%
0.70	spam	4	14	66.7%	100.0%	0.0%
0.60	ham	4	13	66.7%	92.9%	7.1%
0.55	spam	5	13	83.3%	92.9%	7.1%
0.51	ham	5	12	83.3%	85.7%	14.3%
0.49	spam	6	12	100.0%	85.7%	14.3%
0.38	ham	6	11	100.0%	78.6%	21.4%
0.35	ham	6	10	100.0%	71.4%	28.6%
0.30	ham	6	9	100.0%	64.3%	35.7%
0.25	ham	6	8	100.0%	57.1%	42.9%
0.21	ham	6	7	100.0%	50.0%	50.0%
0.20	ham	6	6	100.0%	42.9%	57.1%
0.19	ham	6	5	100.0%	35.7%	64.3%
0.19	ham	6	4	100.0%	28.6%	71.4%
0.18	ham	6	3	100.0%	21.4%	78.6%
0.15	ham	6	2	100.0%	14.3%	85.7%
0.11	ham	6	1	100.0%	7.1%	92.9%
0.10	ham	6	0	100.0%	0.0%	100.0%

Figura 8: La curva ROC traza lo que sucede con la tasa de verdaderos positivos del modelo frente a la tasa de falsos positivos, para conjuntos de ejemplos cada vez más grandes.

Una propiedad importante de las curvas ROC es que no se ven afectadas por el problema del desequilibrio de clases en el que uno de los dos resultados (normalmente la clase positiva) es mucho más raro que el otro.

Muchas métricas de rendimiento, como la precisión, pueden ser engañosas para los datos desequilibrados. Este no es el caso de las curvas ROC, porque ambas dimensiones del gráfico se basan únicamente en las tasas dentro de los valores positivos y negativos y, por lo tanto,

la relación entre los positivos y los negativos no afecta el resultado. Debido a que muchas de las tareas de aprendizaje automático más importantes implican resultados gravemente desequilibrados, las curvas ROC son una herramienta muy útil para comprender la calidad general de un modelo.

Comparación de curvas ROC

Si las curvas ROC son útiles para evaluar un único modelo, no sorprende que también sean útiles para comparar entre modelos. Intuitivamente, sabemos que las curvas más cercanas a la parte superior izquierda del área del gráfico son mejores. En la práctica, la comparación suele ser más complicada que esto, ya que las diferencias entre las curvas suelen ser sutiles en lugar de obvias, y la interpretación es matizada y específica de cómo se utilizará el modelo.

Para entender los matices, comencemos por considerar qué hace que dos modelos tracen curvas diferentes en el gráfico ROC. Comenzando en el origen, la longitud de la curva se extiende a medida que se predice que los ejemplos de conjuntos de prueba adicionales serán positivos. Debido a que el eje y representa la tasa de positivos verdaderos y el eje x representa la tasa de positivos falsos, una trayectoria ascendente más pronunciada es una relación implícita, lo que implica que el modelo es mejor para identificar los ejemplos positivos sin cometer tantos errores.

Esto se ilustra en la figura 9, que representa el comienzo de las curvas ROC para dos modelos imaginarios.

Para la misma cantidad de predicciones (indicada por las longitudes iguales de los vectores que surgen del origen), el primer modelo tiene una tasa de verdaderos positivos más alta y una tasa de falsos positivos más baja, lo que implica que es el que tiene mejor desempeño de los dos:

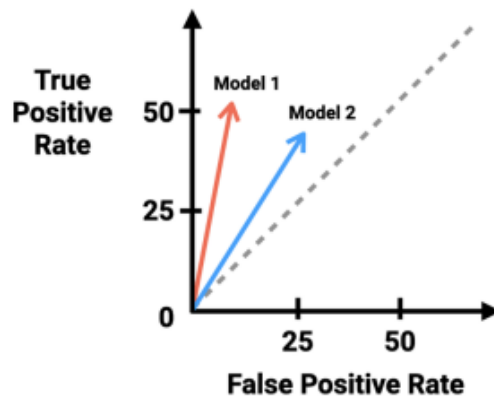


Figura 9: Para la misma cantidad de predicciones, el modelo 1 supera al modelo 2 porque tiene una tasa de verdaderos positivos más alta.

Supongamos que continuamos trazando las curvas ROC para cada uno de estos dos modelos, evaluando las predicciones del modelo en todo el conjunto de datos. En este caso, tal vez el primer modelo continúa superando al segundo en todos los puntos de la curva, como se muestra en la figura 10.

Para todos los puntos de la curva, el primer modelo tiene una tasa de verdaderos positivos más alta y una tasa de falsos positivos más baja, lo que significa que es el que tiene mejor desempeño en todo el conjunto de datos:

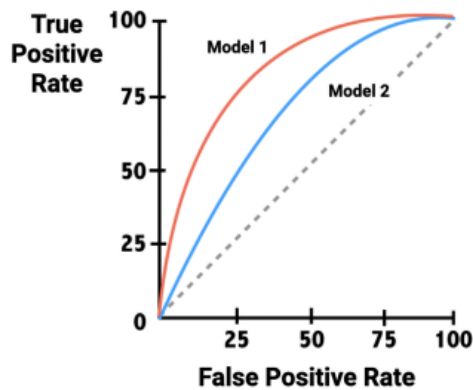


Figura 10: El modelo 1 tiene un mejor desempeño consistente que el modelo 2, con una tasa de verdaderos positivos más alta y una tasa de falsos positivos más baja en todos los puntos de la curva.

Aunque el segundo modelo es claramente inferior en el ejemplo anterior, elegir el que ofrece un mejor rendimiento no siempre es tan fácil. La figura 11 muestra curvas ROC que se cruzan, lo que sugiere que ninguno de los dos modelos ofrece el mejor rendimiento para todas las aplicaciones:

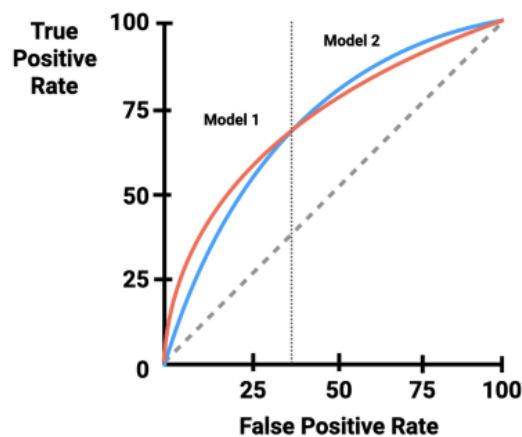


Figura 11: Tanto el modelo 1 como el modelo 2 ofrecen un mejor rendimiento para diferentes subconjuntos de datos.

El punto de intersección entre las dos curvas ROC divide el gráfico en dos áreas: una en la que el primer modelo tiene una tasa de verdaderos positivos más alta y la otra en la que sucede lo contrario. Entonces, ¿cómo sabemos qué modelo es el “mejor” para un caso de uso determinado?

Para responder a esta pregunta, al comparar dos curvas, es útil comprender que ambos modelos intentan ordenar el conjunto de datos en orden de mayor a menor probabilidad de que cada ejemplo sea de la clase positiva. Los modelos que pueden ordenar mejor el conjunto de datos de esta manera tendrán curvas ROC más cerca de la parte superior izquierda del gráfico.

El primer modelo en la figura 11 salta a una ventaja temprana porque pudo ordenar una mayor cantidad de ejemplos positivos en la parte delantera del conjunto de datos, pero después de este aumento inicial, el segundo modelo pudo alcanzar y superar al otro al ordenar de manera lenta y constante los ejemplos positivos frente a los ejemplos negativos en el resto del conjunto de datos. Aunque el segundo modelo puede tener un mejor desempeño general en el conjunto de datos completo, tendemos a preferir modelos que funcionan mejor al principio, los que toman las llamadas “frutas al alcance de la mano” en el conjunto de datos. La justificación para preferir estos modelos es que muchos modelos del mundo real se utilizan solo para la acción en un subconjunto de los datos.

Por ejemplo, considera un modelo utilizado para identificar a los clientes que tienen más probabilidades de responder a una campaña publicitaria por correo directo. Si pudiéramos permitirnos enviar un correo a todos los clientes potenciales, un modelo sería innecesario. Pero como no tenemos el presupuesto para enviar el anuncio a todas las direcciones, el modelo se utiliza para estimar la probabilidad de que el destinatario compre el producto después de ver el anuncio. Un modelo que sea mejor para clasificar a los verdaderos compradores más probables al principio de la lista tendrá una pendiente mayor al principio de la curva ROC y reducirá el presupuesto de marketing necesario para adquirir compradores. En la figura 11, el primer modelo sería más adecuado para esta tarea.

En contraste con este enfoque, otra consideración son los costos relativos de varios tipos de errores; Los falsos positivos y los falsos negativos suelen tener un impacto diferente en el mundo real. Si sabemos que un filtro de spam o una prueba de detección de cáncer deben apuntar a una tasa de verdaderos positivos específica, como el 90 por ciento o el 99 por ciento, favoreceremos el modelo que tenga la tasa de falsos positivos más baja en los niveles deseados.

Aunque ninguno de los dos modelos sería muy bueno debido a la alta tasa de falsos positivos, la figura 11 sugiere que el segundo modelo sería ligeramente preferible para estas aplicaciones.

Como demuestran estos ejemplos, las curvas ROC permiten comparaciones del rendimiento de los modelos que también consideran cómo se utilizarán los modelos. Esta flexibilidad es apreciada en comparación con métricas numéricas más simples como la precisión o kappa, pero también puede ser deseable cuantificar la curva ROC en una única métrica que se pueda comparar cuantitativamente, de forma muy similar a estas estadísticas. La siguiente sección presenta exactamente este tipo de medida.

El área bajo la curva ROC

Comparar las curvas ROC puede ser algo subjetivo y específico del contexto, por lo que siempre se necesitan métricas que reduzcan el rendimiento a un único valor numérico para simplificar y aportar objetividad a la comparación. Si bien puede resultar difícil decir qué hace que una curva ROC sea “buena”, en general, sabemos que cuanto más cerca esté la curva ROC de la parte superior izquierda del gráfico, mejor será para identificar valores positivos. Esto se puede medir utilizando una estadística conocida como el **área bajo la curva ROC** (AUC, *area under the ROC curve*).

El AUC trata el diagrama ROC como un cuadrado bidimensional y mide el área total bajo la curva ROC. El AUC varía de 0.5 (para un clasificador sin valor predictivo) a 1.0 (para un clasificador perfecto). Una convención para interpretar las puntuaciones del AUC utiliza un sistema similar a las calificaciones académicas con letras:

- **A:** Sobresaliente = 0.9 a 1.0
- **B:** Excelente/Bueno = 0.8 a 0.9
- **C:** Aceptable/Regular = 0.7 a 0.8
- **D:** Deficiente = 0.6 a 0.7
- **E:** Sin discriminación = 0.5 a 0.6

Como ocurre con la mayoría de las escalas de este tipo, los niveles pueden funcionar mejor para algunas tareas que para otras; la categorización es algo subjetiva (*difusa, fuzzy*).

Es poco frecuente, pero posible, que la curva ROC caiga por debajo de la diagonal, lo que hace que el AUC sea inferior a 0.50. Esto significa que el clasificador funciona peor que el aleatorio.

Normalmente, esto se debe a un error de codificación, porque un modelo que constantemente hace predicciones incorrectas obviamente ha aprendido algo útil sobre los datos: simplemente está aplicando las predicciones en la dirección incorrecta. Para solucionar este problema, confirma que los casos positivos estén codificados correctamente o simplemente invierte las predicciones de modo que cuando el modelo prediga la clase negativa, elija la clase positiva en su lugar.

Cuando el uso del AUC comenzó a generalizarse, algunos lo trataron como una medida definitiva del rendimiento del modelo, aunque, lamentablemente, esto no es cierto en todos los casos. En términos generales, los valores de AUC más altos reflejan clasificadores que son mejores para clasificar un ejemplo positivo aleatorio en mayor medida que un ejemplo negativo aleatorio. Sin embargo, la figura 12 ilustra el hecho importante de que dos curvas ROC pueden tener formas muy diferentes, pero tener un AUC idéntico:

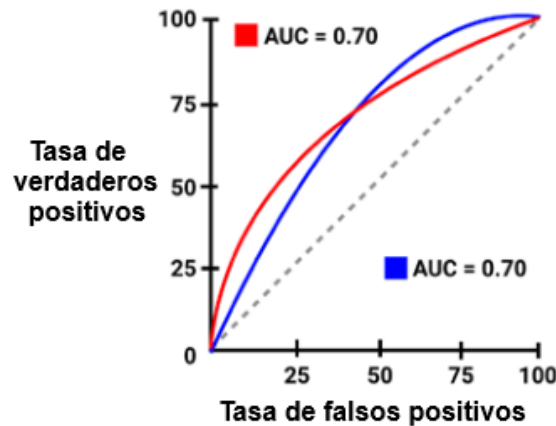


Figura 12: Las curvas ROC pueden tener diferentes rendimientos a pesar de tener el mismo AUC.

Dado que el AUC es una simplificación de la curva ROC, el AUC por sí solo no es suficiente para identificar el “mejor” modelo para todos los casos de uso. La práctica más segura es utilizar el AUC en combinación con un examen cualitativo de la curva ROC, como se describió anteriormente en este documento. Si dos modelos tienen un AUC idéntico o similar, generalmente es preferible elegir el que tenga un mejor rendimiento de manera temprana. Además, incluso en el caso de que un modelo tenga un mejor AUC general, puede preferirse un modelo que tenga una tasa de verdaderos positivos inicial más alta para aplicaciones que utilizarán solo un subconjunto de las predicciones más confiables.

Creación de curvas ROC y cálculo del AUC en R

El paquete pROC proporciona un conjunto de funciones fáciles de usar para crear curvas ROC y calcular el AUC. El sitio web de pROC en <https://web.expasy.org/pROC/>, incluye una lista del conjunto completo de funciones, así como varios ejemplos de las capacidades de visualización. Antes de continuar, asegúrate de haber instalado el paquete utilizando el comando `install.packages("pROC")`.

Para obtener más información sobre el paquete pROC, consulta: pROC: an open-source package for R and S+ to analyze and compare ROC curves, Robin, X, Turck, N, Hainard, A, Tiberti, N, Lisacek, F, Sanchez, JC y Mueller M, BMC Bioinformatics, 2011, pp. 12-77.

Para crear visualizaciones con pROC, se necesitan dos vectores de datos. El primero debe contener la probabilidad estimada de la clase positiva y el segundo debe contener los valores de clase previstos. Para el clasificador SMS, proporcionaremos las probabilidades estimadas de spam y las etiquetas de clase reales a la función `roc()` de la siguiente manera:

```
> library(pROC)
> sms_roc <- roc(sms_results$actual_type, sms_results$prob_spam)
```

Usando el objeto `sms_roc`, podemos visualizar la curva ROC con la función `plot()` de R. Como se muestra en las siguientes líneas de código, se pueden usar muchos de los parámetros estándar para ajustar la visualización, como `main` (para agregar un título), `col` (para cambiar el color de la línea) y `lwd` (para ajustar el ancho de la línea). El parámetro `legacy.axes` indica a pROC que use un eje x de $1 - \text{especificidad}$, que es una convención popular:

```
> plot(sms_roc, main = "ROC curve for SMS spam filter",
+   col = "blue", lwd = 2, legacy.axes = TRUE)
```

El resultado final es un gráfico ROC con una línea de referencia diagonal que representa un clasificador de referencia sin valor predictivo:

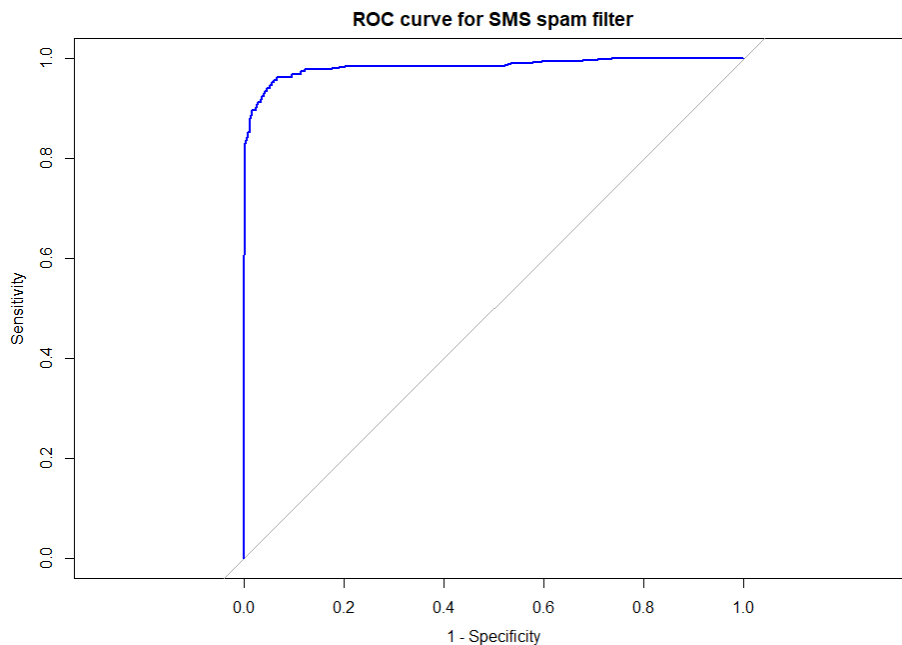


Figura 13: La curva ROC del clasificador SMS Naive Bayes.

En términos cualitativos, podemos ver que esta curva ROC parece ocupar el espacio en la esquina superior izquierda del diagrama, lo que sugiere que está más cerca de un clasificador perfecto que la línea discontinua que representa un clasificador inútil.

Para comparar el rendimiento de este modelo con otros modelos que realizan predicciones en el mismo conjunto de datos, podemos agregar curvas ROC adicionales al mismo gráfico. Supongamos que también hemos entrenado un modelo k-NN en los datos SMS utilizando la función `knn()` descrita en el documento, Aprendizaje perezoso: Clasificación utilizando vecinos más cercanos. Utilizando este modelo, se calcularon las probabilidades predichas de spam para cada registro en el conjunto de prueba y se guardaron en un archivo CSV, que podemos cargar aquí. Después de cargar el archivo, aplicaremos la función `roc()` como antes para calcular la curva ROC, luego usaremos la función `plot()` con el parámetro `add = TRUE` para agregar la curva al gráfico anterior:

```
> # compare to kNN
> sms_results_knn <- read.csv("sms_results_knn.csv")
> sms_roc_knn <- roc(sms_results$actual_type, sms_results_knn$p_spam)
> plot(sms_roc_knn, col = "red", lwd = 2, add = TRUE)
```

La visualización resultante tiene una segunda curva que representa el desempeño del modelo k-NN al realizar predicciones en el mismo conjunto de prueba que el modelo Naive Bayes. La curva para k-NN es consistentemente más baja, lo que sugiere que es un modelo consistentemente peor que el enfoque Naive Bayes:

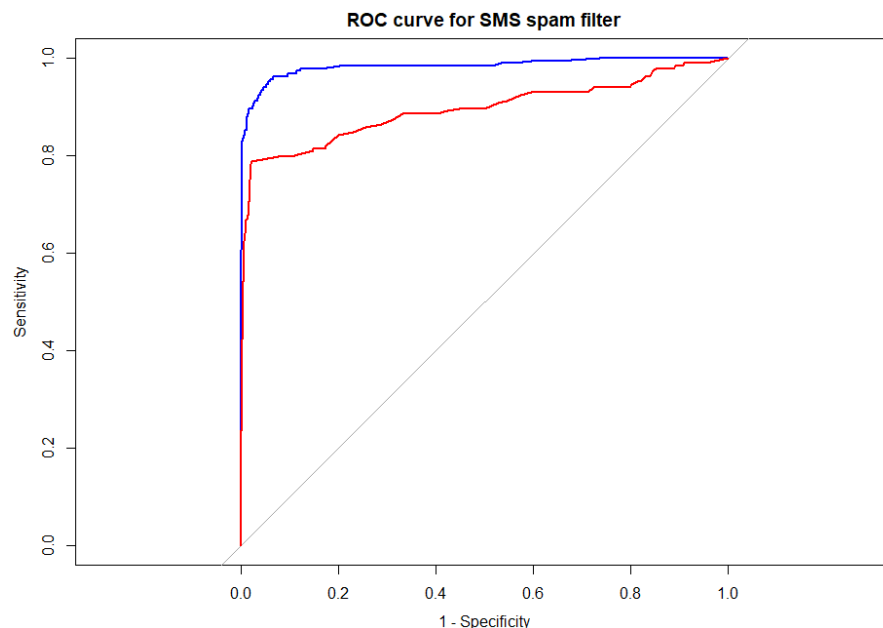


Figura 14: Curvas ROC que comparan el desempeño de Naive Bayes (curva superior) y k-NN en el conjunto de prueba SMS.

Para confirmar esto cuantitativamente, podemos usar el paquete `pROC` para calcular el AUC. Para ello, simplemente aplicamos la función `auc()` del paquete al objeto `sms_roc` de cada modelo, como se muestra en el siguiente código:

```
> auc(sms_roc)
Area under the curve: 0.9836
> auc(sms_roc_knn)
Area under the curve: 0.894
```

El AUC del clasificador SMS Naive Bayes es 0.98, que es extremadamente alto y sustancialmente mejor que el AUC del clasificador k-NN de 0.89. Pero, ¿cómo sabemos si es igualmente probable que el modelo funcione bien en otro conjunto de datos o si la diferencia es mayor de lo esperado por pura casualidad? Para responder a estas preguntas, necesitamos entender mejor hasta qué punto podemos extrapolar las predicciones de un modelo más allá de los datos de prueba.

Esto ya se mencionó antes, pero vale la pena repetirlo: el valor del AUC por sí solo suele ser insuficiente para identificar un “mejor” modelo. En este ejemplo, el AUC identifica el mejor modelo porque las curvas ROC no se cruzan. En otros casos, el “mejor” modelo dependerá de cómo se utilizará el modelo. Cuando las curvas ROC se cruzan, es posible combinarlas en modelos aún más sólidos utilizando técnicas que se tratan más adelante, Mejora del rendimiento del modelo.

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. Sin embargo, el error de resustitución no es un marcador muy útil del rendimiento futuro. 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 ha visto antes. Por este motivo, la tasa de error en los datos de entrenamiento puede ser extremadamente 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 en datos que aún no ha visto. Usamos este método en capítulos anteriores cuando dividimos los datos disponibles en un conjunto para entrenamiento y un conjunto para pruebas. 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, 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 una serie de funciones para estimar el rendimiento futuro. Si está siguiendo los ejemplos de

código R y aún no ha instalado el paquete `caret`, hágalo. También deberá cargar el paquete en la sesión R mediante el comando `library(caret)`.

El método de retención

El procedimiento de dividir los datos en conjuntos de datos de entrenamiento y de prueba que utilizamos en los capítulos anteriores se conoce como el método de retención. Como se muestra en el siguiente diagrama, el conjunto de datos de entrenamiento se utiliza para generar el modelo, que luego se aplica al conjunto de datos de prueba para generar predicciones para la evaluación. Normalmente, alrededor de un tercio de los datos se reserva para la prueba y dos tercios se utilizan para el entrenamiento, pero esta proporción puede variar según la cantidad de datos disponibles. Para garantizar que los conjuntos de datos de entrenamiento y de prueba no tengan diferencias sistemáticas, sus ejemplos se dividen aleatoriamente en los dos grupos.

Figura 10.8: 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 modelo.

Es fácil violar esta regla sin saberlo al elegir un "mejor" modelo en función de los resultados de pruebas repetidas. Por ejemplo, supongamos que construimos 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 elegimos cuidadosamente 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 capítulos 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 válidas 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 usarse 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 10.9: Un conjunto de datos de validación se puede conservar 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 Capítulo 5, *Divide y vencerás: clasificación mediante árboles de decisión y reglas*, para crear conjuntos de datos de entrenamiento y prueba.

Si desea seguir los siguientes ejemplos, descargue el conjunto de datos `credit.csv` del sitio web de Packt Publishing y cárguelo en un marco de datos utilizando el comando `credit <- read.csv("credit.csv")`.

Supongamos que tenemos un marco 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 uno a 1000 utilizando la función `runif()`, que por defecto genera una cantidad específica de valores aleatorios entre cero y uno. La función `runif()` recibe su nombre de la distribución uniforme aleatoria, que se analizó en el Capítulo 2, *Gestión y comprensión de los datos*.

La función `order()` devuelve entonces 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.

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

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 clases son una proporción muy pequeña del conjunto de datos, esto puede provocar que se omitan 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. El código para crear una muestra estratificada de datos de entrenamiento y prueba para el conjunto de datos de crédito se muestra en los siguientes comandos.

Para utilizar la función, se debe especificar un vector de valores de clase (aquí, predeterminado 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:

El vector `in_train` indica los números de fila incluidos en la muestra de entrenamiento. Podemos usar estos números de fila para seleccionar ejemplos para el marco de datos `credit_train`. De manera similar, al usar un símbolo negativo, podemos usar 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 con conjuntos de datos más grandes generalmente tienen un mejor rendimiento, una práctica común es volver a entrenar el modelo con 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 con 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 de *k*-fold (*k*-fold CV), 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 CV de *k*-fold divide aleatoriamente los datos en *k* particiones aleatorias completamente separadas llamadas pliegues.

Aunque *k* se puede establecer en cualquier número, la convención más común es usar CV 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 CV de k-fold es el método leave-one-out, que realiza CV 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:

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()`:

Aquí, vemos que el primer fold se llama `Fold01` y almacena 100 números enteros que indican las 100 filas en el marco 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:

Para realizar el CV 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 CV de 10 pliegues. Primero, necesitamos cargar algunos paquetes de R:

`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 desea, puede 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:

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 marco 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()`:

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

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

Esta estadística kappa es bastante baja, lo que corresponde a "justo" en la escala de interpretación, lo que sugiere que el modelo de calificación crediticia funciona solo marginalmente mejor que el azar. En el próximo capítulo, examinaremos métodos automatizados basados en CV de 10 veces que pueden ayudarnos a mejorar el rendimiento de este modelo.

Quizás el método estándar actual para estimar de manera confiable el rendimiento del modelo sea el CV de k veces repetido. Como puede adivinar por el nombre, esto implica aplicar repetidamente el CV de k veces y promediar los resultados. Una estrategia común es realizar el CV de 10 veces 10 veces.

Aunque requiere mucho trabajo computacional, esto proporciona una estimación muy sólida.

Muestreo bootstrap

Una alternativa un poco menos popular pero aún bastante utilizada al CV 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 CV 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. Luego, 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.

Si se utiliza el muestreo con reemplazo, como se describió anteriormente, la probabilidad de que cualquier instancia dada se incluya en el conjunto de datos de entrenamiento es del 63,2 por ciento. En consecuencia, la probabilidad de que cualquier instancia esté en el conjunto de datos de prueba es del 36,8 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 CV de 10 veces, que utiliza el 90 por ciento de los ejemplos para el entrenamiento, la muestra de bootstrap es menos representativa del conjunto de datos completo.

Debido a que es probable que un modelo entrenado con solo el 63,2 por ciento de los datos de entrenamiento tenga un rendimiento peor que un modelo entrenado con un conjunto de entrenamiento más grande, las estimaciones de rendimiento del bootstrap pueden ser sustancialmente inferiores a 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 rendimiento final como una función del rendimiento tanto en los datos de entrenamiento (que son demasiado optimistas) como en los datos de prueba (que son demasiado pesimistas). La tasa de error final se calcula entonces como:

Una ventaja del muestreo bootstrap sobre la validación cruzada es que tiende a funcionar mejor con conjuntos de datos muy pequeños. Además, el muestreo bootstrap tiene aplicaciones más allá de la medición del rendimiento. En particular, en el siguiente capítulo aprenderemos cómo se pueden utilizar los principios del muestreo bootstrap para mejorar el rendimiento del modelo.

Resumen

En este capítulo 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. Una serie de medidas basadas en matrices de confusión capturan mejor el equilibrio entre los costos de varios tipos de errores. 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 rendimiento de un modelo es considerar qué tan bien cumple, o no, otros objetivos. Por ejemplo, es posible que deba explicar la lógica de un modelo en un lenguaje sencillo, 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.

Una extensión obvia de la medición del rendimiento es identificar formas automatizadas de encontrar los "mejores" modelos para una tarea en particular. En el próximo capítulo, nos basaremos en 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.