Muller - Guido I

Outline breve (págs. 265-268)

- **Paso de la teoría a la práctica de evaluación**
    - Se abre el Cap. 5 (“Evaluación y mejora del modelo”) aclarando que el foco será supervisado; la calidad se juzgará por la capacidad de generalizar, no por el ajuste al train set.
- **Revisión del esquema *train/test split***
    - Ejemplo sintético con `LogisticRegression` y `train_test_split`: código de demostración y score ≈ 0.88 para ilustrar la métrica por defecto en clasificación.
- **Validación cruzada (k-fold)**
    - Descripción paso a paso de la división en *k* pliegues (fig. 5-1); valores habituales *k*=5 ó 10.
    - Objetivo: estimar la distribución de la métrica y reducir la varianza que surge de una sola partición aleatoria.
- **Implementación con `cross_val_score`**
    - Uso básico sobre el dataset Iris; salida de los cinco scores y cálculo de la media (≈ 0.96).
    - Parámetro `cv` controla el número de folds; el rango de puntuaciones revela sensibilidad del modelo a la selección de datos.
- **Costes y beneficios**
    - Ventaja: se entrena con hasta 90 % de los datos (en 10-fold) y cada muestra actúa una vez como test.
    - Desventaja: coste computacional multiplicado por *k*.

Muller - Guido II

Outline breve (págs. 289 – 317)

- **Por qué la métrica importa**
    - Antes de comparar modelos, el libro recuerda que la métrica debe reflejar el objetivo final (p. ej. coste de un falso negativo en cáncer) y advierte que accuracy rara vez basta en problemas reales.

Métricas para clasificación binaria

1. **Confusion matrix como punto de partida**
    - Desglose TP-FP-TN-FN y ejemplo “9 vs resto” en dígitos; accuracy 90 % puede no significar nada si el predictor siempre elige la clase mayoritaria.
2. **Precision, Recall y F1**
    - Cuándo optimizar cada una (evitar falsos positivos vs. falsos negativos) y cómo F1 resume su equilibrio; diferencias claras entre modelos que parecían equivalentes en accuracy.
3. **Desequilibrio de clases**
    - Uso de `DummyClassifier` para mostrar que un modelo “tonto” puede alcanzar alta accuracy; necesidad de métricas sensibles a clase minoritaria.

Curvas y ajustes de umbral

- **Precision-Recall curve**
    - `precision_recall_curve` revela todas las compensaciones posibles; fijar un “punto de operación” (ej. 90 % recall) según requisitos de negocio.
- **ROC curve y AUC**
    - Definición de FPR/TPR, interpretación de la esquina superior-izquierda y comparación SVM vs Random Forest; área bajo la curva como resumen único (AUC).
    - Ejemplo con tres SVM de distinto γ: misma accuracy, AUC muy diferente, mostrando la utilidad de la métrica.
- **Ajuste explícito de umbrales**
    - Cómo mover el umbral de `decision_function`/`predict_proba` para privilegiar recall o precisión según la aplicación.

Métricas más allá de la clasificación binaria

- **Multiclase**
    - Promedios macro/micro/weighted derivan de las métricas binarias; elección depende de si se priorizan clases o muestras por igual.
- **Regresión**
    - Se mencionan R^2 (scikit-learn default), MSE y MAE como alternativas frecuentes cuando las decisiones dependen de errores absolutos o cuadrados.

Cierre operativo

- **Selección de modelos con grid search + AUC**
    - Demostración de cómo cambiar el parámetro `scoring` altera qué solución se considera “mejor” y puede descubrir hiperparámetros que accuracy ignoraría.

El tramo se concentra en pasar de “¿qué tan bien acierta?” a “¿qué métrica refleja el riesgo real?”, cubriendo herramientas prácticas (curvas, AUC, F1) y la adaptación de scikit-learn para optimizar el indicador adecuado.

Geron

Outline breve (págs. 251 – 256)

- **Comparación de complejidad en regresión polinómica**
    - Fig. 4-14: modelo lineal **subajusta**, polinomio de 300° **sobreajusta** y polinomio cuadrático encaja mejor; plantea el problema de decidir cuán complejo debe ser el modelo cuando se desconoce la función generadora.
- **Curvas de aprendizaje**
    - Gráficos del error de entrenamiento y validación a lo largo de la iteración o del tamaño del set; permiten diagnosticar sobreajuste (gran brecha entre curvas) y subajuste (curvas altas y cercanas). Se generan fácilmente con `learning_curve()` de Scikit-Learn, que admite entrenamiento incremental con `exploit_incremental_learning=True`.
- **Ejemplo 1 – Regresión lineal simple**
    - Fig. 4-15: ambas curvas se estancan alto y próximas ⇒ **subajuste**; añadir más datos no ayuda, se necesita un modelo más rico o mejores características.
- **Ejemplo 2 – Regresión polinomial grado 10**
    - Fig. 4-16: error de entrenamiento muy bajo, error de validación sustancialmente mayor ⇒ **sobreajuste**. El consejo práctico: incorporar más instancias puede cerrar la brecha entre curvas.
- **Sesgo-varianza y error irreducible**
    - El error de generalización se descompone en **sesgo² + varianza + irreducible**:
        - *Sesgo*: suposiciones demasiado simples (ej. modelo lineal para datos curvos).
        - *Varianza*: sensibilidad excesiva a pequeñas variaciones del set (modelos muy flexibles).
        - *Error irreducible*: ruido inherente; se mitiga limpiando datos, no con el modelo.
    - Incrementar la complejidad eleva la varianza y reduce el sesgo; disminuirla hace lo contrario—de ahí la “compensación sesgo-varianza”.
- **Recomendaciones prácticas**

- *Subajuste* → usar modelos más complejos o diseñar mejores features.
- *Sobreajuste* → obtener más datos o aplicar regularización; seguir las curvas de aprendizaje para monitorizar el efecto.

Alppaydin

Outline breve (págs. 116 – 120)

- **Errores y R^2 en regresión paramétrica:** define el error cuadrático relativo (ERSE) y el coeficiente de determinación R^2 para decidir si el modelo supera al simple promedio.
- **Descomposición sesgo-varianza:** separa el error esperado en ruido, sesgo y varianza, planteando el dilema de equilibrar complejidad del modelo y complejidad de los datos.
- **Ejemplo con polinomios (orden 1-5):** ilustra cómo más complejidad reduce sesgo pero eleva varianza y riesgo de sobreajuste (figs. 4.5-4.6).
- **Validación cruzada para elegir orden óptimo:** muestra el “codo” donde el error de validación deja de mejorar y sugiere CV como procedimiento práctico de selección.
- **Regularización (E + λ·complejidad):** introduce la penalización directa de modelos complejos, con λ ajustado por CV para bajar varianza a costa de algo de sesgo. (no entra)
- **Mención a combinaciones de modelos:** anticipa técnicas de promediado (bagging, boosting) como otra vía para reducir varianza, tratadas en capítulos posteriores. (no entra)

CAPÍTULO 5

Evaluación y mejora del modelo

Después de analizar los fundamentos del aprendizaje supervisado y no supervisado, y de explorar una variedad de algoritmos de aprendizaje automático, ahora profundizaremos en la evaluación de modelos y la selección de parámetros.

Nos centraremos en los métodos supervisados, regresión y clasificación, ya que evaluar y seleccionar modelos en el aprendizaje no supervisado es a menudo un proceso muy cualitativo (como vimos en el Capítulo 3).

Para evaluar nuestros modelos supervisados, hasta ahora hemos dividido nuestro conjunto de datos en un conjunto de entrenamiento y un conjunto de prueba mediante la función train_test_split , hemos creado un modelo en el conjunto de entrenamiento mediante el método de ajuste y lo hemos evaluado en el conjunto de prueba mediante el método de puntuación , que calcula la fracción de muestras correctamente clasificadas para la clasificación. A continuación, se muestra un ejemplo de este proceso:

En[2]:

```
desde sklearn.datasets importar make_blobs desde
sklearn.linear_model importar LogisticRegression desde sklearn.model_selection
importar train_test_split

# crear un conjunto de datos sintéticos X,
y = make_blobs(random_state=0) # dividir los
datos y las etiquetas en un conjunto de entrenamiento y uno de prueba X_train,
X_test, y_train, y_test = train_test_split(X, y, random_state=0) # crear una instancia de un modelo y ajustarlo
al conjunto de entrenamiento logreg = LogisticRegression().fit(X_train, y_train)
# evaluar el modelo en el conjunto de prueba print("Test set scoring:
{:.2f}".format(logreg.score(X_test, y_test)))
```

Salida[2]:

```
Puntuación del conjunto de pruebas: 0,88
```

Recuerde, la razón por la que dividimos nuestros datos en conjuntos de entrenamiento y prueba es que estamos interesados en medir qué tan bien nuestro modelo se generaliza a datos nuevos, nunca antes vistos.
No nos interesa qué tan bien se ajusta nuestro modelo al conjunto de entrenamiento, sino qué tan bien puede realizar predicciones para datos que no se observaron durante el entrenamiento.

En este capítulo, ampliaremos dos aspectos de esta evaluación. Primero, presentaremos la validación cruzada, una forma más robusta de evaluar el rendimiento de la generalización, y analizaremos métodos para evaluar el rendimiento de la clasificación y la regresión que van más allá de las medidas predeterminadas de precisión y R.
^{2} proporcionada por el método de puntuación .

También discutiremos la búsqueda en cuadrícula, un método efectivo para ajustar los parámetros en modelos supervisados para obtener el mejor rendimiento de generalización.

Validación cruzada

La validación cruzada es un método estadístico para evaluar el rendimiento de la generalización, más estable y exhaustivo que dividir los datos en conjuntos de entrenamiento y prueba. En la validación cruzada, los datos se dividen repetidamente y se entrenan múltiples modelos. La versión más común de validación cruzada es la validación cruzada de k-folds, donde k es un número especificado por el usuario, generalmente 5 o 10. Al realizar una validación cruzada de cinco-folds, los datos se dividen primero en cinco partes de tamaño (aproximadamente) igual, denominadas "folds".

A continuación, se entrena una secuencia de modelos. El primer modelo se entrena utilizando el primer pliegue como conjunto de prueba, y los pliegues restantes (2-5) se utilizan como conjunto de entrenamiento. El modelo se construye utilizando los datos de los pliegues 2-5, y luego se evalúa la precisión en el pliegue 1. A continuación, se construye otro modelo, esta vez utilizando el pliegue 2 como conjunto de prueba y los datos de los pliegues 1, 3, 4 y 5 como conjunto de entrenamiento. Este proceso se repite utilizando los pliegues 3, 4 y 5 como conjuntos de prueba.
Para cada una de estas cinco divisiones de datos en conjuntos de entrenamiento y de prueba, calculamos la precisión.
Finalmente, hemos recopilado cinco valores de precisión. El proceso se ilustra en la Figura 5-1:

En[3]:

```
mglearn.plots.plot_cross_validation()
```

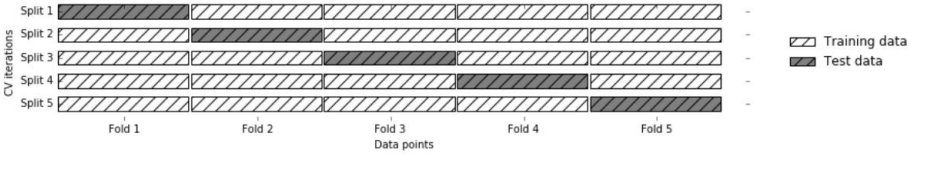

Figura 5-1. División de datos en la validación cruzada quíntuple

Generalmente, la primera quinta parte de los datos es el primer pliegue, la segunda quinta parte de los datos es el segundo pliegue, y así sucesivamente.

Validación cruzada en scikit-learn

La validación cruzada se implementa en scikit-learn mediante la función cross_val_score del módulo model_selection . Los parámetros de la función cross_val_score son el modelo que queremos evaluar, los datos de entrenamiento y las etiquetas de la verdad fundamental. Evalúemos LogisticRegression en el conjunto de datos iris :

En[4]:

```
desde sklearn.model_selection importar cross_val_score desde
sklearn.datasets importar load_iris desde
sklearn.linear_model importar LogisticRegression

iris = cargar_iris() logreg =
Regresión logística()

puntuaciones = cross_val_score(logreg, iris.data, iris.target) print("Puntuaciones de
validación cruzada: {}".format(scores))
```

Salida[4]:

```
Puntuaciones de validación cruzada: [ 0,961 0,922 0,958]
```

De forma predeterminada, cross_val_score realiza una validación cruzada triple y devuelve tres valores de precisión. Podemos cambiar el número de veces utilizadas modificando el parámetro cv. ter:

En[5]:

```
puntuaciones = cross_val_score(logreg, iris.data, iris.target, cv=5) print("Puntuaciones de
validación cruzada: {}".format(scores))
```

Salida[5]:

```
Puntuaciones de validación cruzada: [ 1.        0,967 0,933 0,9         1.        ]
```

Una forma común de resumir la precisión de la validación cruzada es calcular la media:

En[6]:

```
print("Puntuación promedio de validación cruzada: {:.2f}".format(scores.mean()))
```

Salida[6]:

```
Puntuación media de validación cruzada: 0,96
```

Utilizando la validación cruzada media, podemos concluir que esperamos que el modelo tenga una precisión promedio de alrededor del 96 %. Al observar las cinco puntuaciones obtenidas mediante la validación cruzada quíntuple, también podemos concluir que existe una varianza relativamente alta en la precisión entre los pliegues, que oscila entre el 100 % y el 90 %. Esto podría implicar que el modelo depende en gran medida de los pliegues específicos utilizados para el entrenamiento, o también podría deberse simplemente al pequeño tamaño del conjunto de datos.

Beneficios de la validación cruzada

Existen varias ventajas al usar la validación cruzada en lugar de una única división en un conjunto de entrenamiento y uno de prueba. Primero, recuerde que train_test_split realiza una división aleatoria de los datos. Imaginemos que, al dividir los datos aleatoriamente, tenemos suerte y todos los ejemplos difíciles de clasificar terminan en el conjunto de entrenamiento. En ese caso, el conjunto de prueba solo contendrá ejemplos fáciles, y la precisión de dicho conjunto será excesivamente alta. Por el contrario, si no tenemos suerte, podríamos haber incluido aleatoriamente todos los ejemplos difíciles de clasificar en el conjunto de prueba y, en consecuencia, haber obtenido una puntuación excesivamente baja.
Sin embargo, al usar la validación cruzada, cada ejemplo estará en el conjunto de entrenamiento exactamente una vez: cada ejemplo está en uno de los pliegues, y cada pliegue es el conjunto de prueba una vez. Por lo tanto, el modelo debe generalizarse correctamente a todas las muestras del conjunto de datos para que todas las puntuaciones de validación cruzada (y su media) sean altas.

Tener múltiples divisiones de los datos también proporciona información sobre la sensibilidad de nuestro modelo a la selección del conjunto de datos de entrenamiento. Para el conjunto de datos de iris , observamos precisiones de entre el 90 % y el 100 %. Este rango es considerable y nos da una idea del rendimiento del modelo en el peor y el mejor de los casos al aplicarlo a nuevos datos.

Otra ventaja de la validación cruzada, en comparación con una sola división de los datos, es que los utilizamos de forma más eficaz. Al usar train_test_split, solemos usar el 75 % de los datos para el entrenamiento y el 25 % para la evaluación. Al usar la validación cruzada quíntuple, en cada iteración podemos usar cuatro quintos de los datos (80 %) para ajustar el modelo. Al usar la validación cruzada décima, podemos usar nueve décimos de los datos (90 %) para ajustar el modelo. Un mayor número de datos suele resultar en modelos más precisos.

La principal desventaja de la validación cruzada es el mayor coste computacional. Dado que ahora entrenamos k modelos en lugar de uno solo, la validación cruzada será aproximadamente k veces más lenta que realizar una sola división de los datos.

Es importante tener en cuenta que la validación cruzada no permite crear un modelo aplicable a nuevos datos. No devuelve un modelo. Al llamar a cross_val_score , se crean varios modelos internamente, pero el propósito de la validación cruzada es únicamente evaluar la generalización de un algoritmo dado al entrenarlo con un conjunto de datos específico.

Validación cruzada estratificada de k-folds y otras estrategias. Dividir el

conjunto de datos en k-folds comenzando con la primera parte k-ésima de los datos, como se describió en la sección anterior, podría no ser siempre una buena idea. Por ejemplo, veamos el conjunto de datos iris :

El modelo que utiliza una configuración de parámetros particular en una división de validación cruzada particular se puede realizar de forma completamente independiente de las otras configuraciones de parámetros y modelos. Esto hace que la búsqueda en cuadrícula y la validación cruzada sean ideales para la paralelización en múltiples núcleos de CPU o en un clúster. Puede usar múltiples núcleos en Grid SearchCV y cross_val_score configurando el parámetro n_jobs con el número de núcleos de CPU que desea usar. Puede configurar n_jobs=-1 para usar todos los núcleos disponibles.

Debe tener en cuenta que scikit-learn no permite la anidación de operaciones paralelas.
Por lo tanto, si usa la opción n_jobs en su modelo (por ejemplo, un bosque aleatorio), no podrá usarla en GridSearchCV para buscar en él. Si su conjunto de datos y modelo son muy grandes, es posible que el uso de muchos núcleos consuma demasiada memoria, por lo que debería supervisar el uso de memoria al crear modelos grandes en paralelo.

También es posible paralelizar la búsqueda en cuadrícula y la validación cruzada en varias máquinas de un clúster, aunque al momento de escribir esto, scikit-learn no lo admite. Sin embargo, es posible usar el framework paralelo de IPython para búsquedas en cuadrícula paralelas, si no le importa escribir el bucle for sobre los parámetros, como hicimos en «Búsqueda simple en cuadrícula» en la página 261.

Para los usuarios de Spark, también existe el recientemente desarrollado spark-sklearn paquete, que permite ejecutar una búsqueda de cuadrícula en un clúster Spark ya establecido.

Métricas de evaluación y puntuación

Hasta ahora, hemos evaluado el rendimiento de la clasificación mediante la precisión (la fracción de muestras correctamente clasificadas) y el rendimiento de la regresión utilizando R, solo dos de ^{2}. Sin embargo, estos son las muchas maneras posibles de resumir el rendimiento de un modelo supervisado en un conjunto de datos determinado. En la práctica, estas métricas de evaluación podrían no ser adecuadas para su aplicación, y es importante elegir la métrica correcta al seleccionar entre modelos y ajustar parámetros.

Tenga en mente el objetivo final. Al

seleccionar una métrica, siempre debe tener en mente el objetivo final de la aplicación de aprendizaje automático. En la práctica, generalmente nos interesa no solo hacer predicciones precisas, sino también usar estas predicciones como parte de un proceso de toma de decisiones más amplio. Antes de elegir una métrica de aprendizaje automático, debe pensar en el objetivo general de la aplicación, a menudo llamado métrica de negocio. Las consecuencias de elegir un algoritmo particular para una aplicación de aprendizaje automático son

llamado el impacto empresarial.Quizás el objetivo principal sea evitar accidentes de tráfico o reducir el número de ingresos hospitalarios. También podría ser conseguir más usuarios para tu sitio web o que gasten más dinero en tu tienda. Al elegir un modelo o ajustar parámetros, debes elegir los valores del modelo o parámetro que tengan la mayor influencia positiva en la métrica de negocio. Esto suele ser difícil, ya que evaluar el impacto empresarial de un modelo en particular puede requerir su puesta en producción en un sistema real.

En las primeras etapas del desarrollo, y para ajustar parámetros, a menudo no es factible poner modelos en producción solo para fines de prueba, debido a los altos riesgos comerciales o personales que pueden estar involucrados. Imagine evaluar las capacidades de evasión de peatones de un automóvil autónomo simplemente dejándolo conducir, sin verificarlo primero; ¡si su modelo es malo, los peatones estarán en problemas! Por lo tanto, a menudo necesitamos encontrar algún procedimiento de evaluación sustituto, utilizando una métrica de evaluación que sea más fácil de calcular. Por ejemplo, podríamos probar la clasificación de imágenes de peatones contra no peatones y medir la precisión. Tenga en cuenta que esto es solo un sustituto, y vale la pena encontrar la métrica más cercana al objetivo comercial original que sea factible de evaluar. Esta métrica más cercana debe usarse siempre que sea posible para la evaluación y selección del modelo. El resultado de esta evaluación podría no ser un número único (la consecuencia de su algoritmo podría ser que usted tenga un 10 % más de clientes, pero cada cliente gastará un 15 % menos), pero debería capturar el impacto comercial esperado de elegir un modelo sobre otro.

En esta sección, primero analizaremos las métricas para el importante caso especial de la clasificación binaria, luego pasaremos a la clasificación multiclase y, finalmente, a la regresión.

Métricas para la clasificación binaria. La

clasificación binaria es posiblemente la aplicación más común y conceptualmente sencilla del aprendizaje automático en la práctica. Sin embargo, existen varias advertencias al evaluar incluso esta sencilla tarea. Antes de profundizar en métricas alternativas, veamos cómo la medición de la precisión puede ser engañosa. Recuerde que, para la clasificación binaria, solemos hablar de una clase positiva y una negativa, entendiendo que la clase positiva es la que buscamos.

Tipos de errores

A menudo, la precisión no es una buena medida del rendimiento predictivo, ya que la cantidad de errores que cometemos no contiene toda la información que nos interesa. Imagine una aplicación para la detección temprana del cáncer mediante una prueba automatizada. Si

2 Pedimos a los lectores con mentalidad científica que disculpen el lenguaje comercial de esta sección. No perder de vista el objetivo final es igualmente importante en la ciencia, aunque los autores desconocen que se utilice una frase similar a «impacto empresarial» en ese ámbito.

Si la prueba es negativa, se asumirá que el paciente está sano, mientras que si es positiva, se le realizarán pruebas adicionales. En este caso, una prueba positiva (una indicación de cáncer) se denominaría clase positiva, y una prueba negativa, clase negativa. No podemos asumir que nuestro modelo siempre funcionará a la perfección, ya que cometerá errores. Para cualquier aplicación, debemos preguntarnos cuáles podrían ser las consecuencias de estos errores en el mundo real.

Un posible error es que un paciente sano sea clasificado como positivo, lo que conlleva pruebas adicionales. Esto conlleva algunos costos e inconvenientes para el paciente (y posiblemente algo de angustia mental). Una predicción positiva incorrecta se llama falso positivo. El otro posible error es que un paciente enfermo sea clasificado como negativo y no reciba más pruebas ni tratamiento. El cáncer no diagnosticado podría conducir a problemas de salud graves e incluso podría ser fatal. Un error de este tipo (una predicción negativa incorrecta) se llama falso negativo. En estadística, un falso positivo también se conoce como error de tipo I y un falso negativo como error de tipo II. Nos ceñiremos a "falso negativo" y "falso positivo", ya que son más explícitos y fáciles de recordar. En el ejemplo del diagnóstico de cáncer, está claro que queremos evitar los falsos negativos tanto como sea posible, mientras que los falsos positivos pueden verse como una molestia menor.

Si bien este es un ejemplo particularmente drástico, las consecuencias de los falsos positivos y los falsos negativos rara vez son las mismas. En aplicaciones comerciales, podría ser posible asignar valores monetarios a ambos tipos de errores, lo que permitiría medir el error de una predicción específica en dólares, en lugar de la precisión. Esto podría ser mucho más significativo para tomar decisiones comerciales sobre qué modelo utilizar.

Conjuntos de datos desequilibrados

Los tipos de error juegan un papel importante cuando una de dos clases es mucho más frecuente que la otra. Esto es muy común en la práctica; un buen ejemplo es la predicción de clics, donde cada punto de datos representa una "impresión", un elemento mostrado a un usuario. Este elemento puede ser un anuncio, una historia relacionada o una persona relacionada a la que seguir en una red social. El objetivo es predecir si, al mostrarse un elemento en particular, un usuario hará clic en él (indicando su interés). La mayoría de los elementos que se muestran a los usuarios en internet (en particular, los anuncios) no generan un clic. Podría ser necesario mostrar a un usuario 100 anuncios o artículos para que encuentre algo lo suficientemente interesante como para hacer clic. Esto da como resultado un conjunto de datos donde por cada 99 puntos de datos "sin clic", hay 1 punto de datos "en el que se hizo clic"; en otras palabras, el 99 % de las muestras pertenecen a la clase "sin clic". Los conjuntos de datos en los que una clase es mucho más frecuente que la otra se suelen denominar conjuntos de datos desequilibrados o conjuntos de datos con clases desequilibradas. En realidad, los datos desequilibrados son la norma y es raro que los eventos de interés tengan una frecuencia igual o incluso similar en los datos.

Ahora digamos que construyes un clasificador que tiene un 99 % de precisión en la tarea de predicción de clics. ¿Qué te dice eso? Una precisión del 99 % suena impresionante, pero esto no lo hace.

Se considera el desequilibrio de clases. Se puede lograr una precisión del 99 % sin crear un modelo de aprendizaje automático, prediciendo siempre "sin clic". Por otro lado, incluso con datos desequilibrados, un modelo con una precisión del 99 % podría ser bastante bueno. Sin embargo, la precisión no nos permite distinguir el modelo de "sin clic" constante de un modelo potencialmente bueno.

Para ilustrarlo, crearemos un conjunto de datos desequilibrado 9:1 a partir del conjunto de datos de dígitos , clasificando el dígito 9 frente a las otras nueve clases:

En[37]:

```
desde sklearn.datasets importar load_digits

dígitos = cargar_dígitos() y
= dígitos.objetivo == 9

X_train, X_test, y_train, y_test = train_test_split( dígitos.datos, y,
      estado_aleatorio=0)
```

Podemos usar el DummyClassifier para predecir siempre la clase mayoritaria (aquí “no nueve”) para ver cuán poco informativa puede ser la precisión:

En[38]:

```
de sklearn.dummy importar DummyClassifier
dummy_majority = DummyClassifier(strategy='most_frequent').fit(X_train, y_train) pred_most_frequent =
dummy_majority.predict(X_test) imprimir("Etiquetas predichas
únicas: {}".format(np.unique(pred_most_frequent))) imprimir(" Puntuación de la prueba:
{:.2f}".format(dummy_majority.score(X_test, y_test)))
```

Salida[38]:

```
Etiquetas únicas predichas: [Falso]
Puntuación de la prueba: 0,90
```

Obtuvimos una precisión cercana al 90 % sin aprender nada. Puede parecer sorprendente, pero piénsenlo un momento. Imaginen que alguien les dice que su modelo tiene una precisión del 90 %. Podrían pensar que hizo un excelente trabajo. Pero, dependiendo del problema, ¡podría ser posible con solo predecir una clase! Comparemos esto con el uso de un clasificador real:

En[39]:

```
de sklearn.tree importar DecisionTreeClassifier tree =
DecisionTreeClassifier(max_depth=2).fit(X_train, y_train) pred_tree =
tree.predict(X_test) imprimir(" Puntuación
de la prueba: {:.2f}".format(tree.score(X_test, y_test)))
```

Salida[39]:

```
Puntuación de la prueba: 0,92
```

En cuanto a la precisión, el DecisionTreeClassifier es solo ligeramente mejor que el predictor constante. Esto podría indicar que algo falla en el uso del DecisionTreeClassifier o que la precisión no es una buena medida en este caso.

Para fines de comparación, evaluemos dos clasificadores más, LogisticRegression y el DummyClassifier predeterminado , que hace predicciones aleatorias pero produce clases con las mismas proporciones que en el conjunto de entrenamiento:

En[40]:

```
de sklearn.linear_model importar LogisticRegression

ficticio = DummyClassifier().fit(X_train, y_train) pred_dummy =
dummy.predict(X_test) print("puntuación
ficticia: {:.2f}".format(dummy.score(X_test, y_test)))

logreg = Regresión logística (C=0.1).fit(X_train, y_train) pred_logreg =
logreg.predict(X_test) print("puntuación logreg:
{:.2f}".format(logreg.score(X_test, y_test)))
```

Salida[40]:

```
Puntuación ficticia:
0,80 Puntuación logreg: 0,98
```

El clasificador ficticio que produce un resultado aleatorio es claramente el peor de todos (según la precisión), mientras que LogisticRegression produce muy buenos resultados. Sin embargo, incluso el clasificador aleatorio ofrece una precisión superior al 80 %. Esto dificulta mucho determinar cuál de estos resultados es realmente útil. El problema radica en que la precisión es una medida inadecuada para cuantificar el rendimiento predictivo en este contexto desequilibrado. En el resto de este capítulo, exploraremos métricas alternativas que proporcionen una mejor orientación para la selección de modelos. En particular, nos gustaría contar con métricas que nos indiquen la mejora de un modelo en comparación con las predicciones «más frecuentes» o aleatorias, tal como se calculan en pred_most_frequent y pred_dummy. Si utilizamos una métrica para evaluar nuestros modelos, esta debería ser capaz de descartar estas predicciones sin sentido.

Matrices de confusión

Una de las formas más completas de representar el resultado de la evaluación de la clasificación binaria es mediante matrices de confusión. Analicemos las predicciones de LogisticRegression de la sección anterior utilizando la función confusion_matrix . Ya almacenamos las predicciones en el conjunto de prueba en pred_logreg:

En[41]:

```
desde sklearn.metrics importar confusion_matrix

confusión = matriz_confusión(prueba_y, pred_logreg) print(" Matriz de
confusión:\n{}".format(confusión))
```

Salida[41]:

```
Matriz de confusión:
[[401 2] [ 8 39]]
```

La salida de confusion_matrix es una matriz de dos por dos, donde las filas corresponden a las clases reales y las columnas a las predichas. Cada entrada cuenta la frecuencia con la que una muestra perteneciente a la clase correspondiente a la fila (en este caso, "no nueve" y "nueve") se clasificó como la clase correspondiente a la columna. La siguiente gráfica (Figura 5-10) ilustra este significado:

En[42]:

```
mglearn.plots.plot_confusion_matrix_illustration()
```

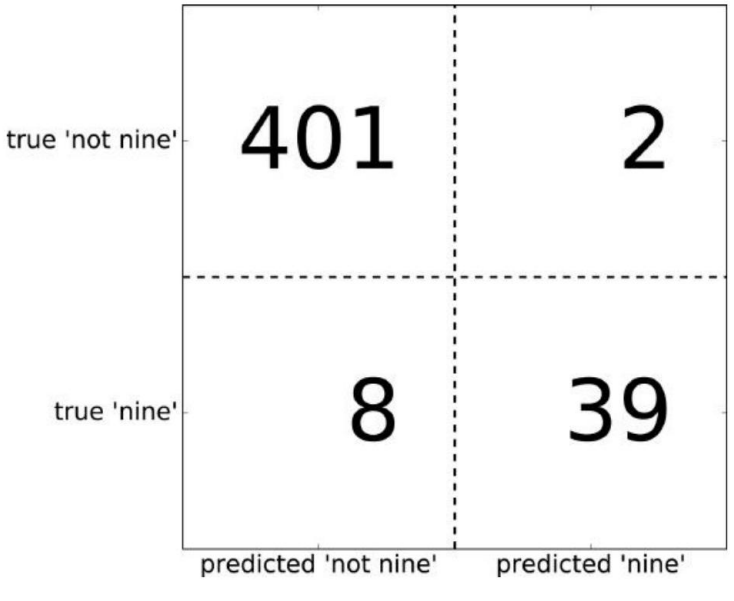

Figura 5-10. Matriz de confusión de la tarea de clasificación «nueve vs. resto»

Las entradas en la diagonal principal3 de la matriz de confusión corresponden a clasificaciones correctas, mientras que otras entradas nos indican cuántas muestras de una clase se clasificaron erróneamente como otra clase.

Si declaramos "un nueve" como la clase positiva, podemos relacionar las entradas de la matriz de confusión con los términos falso positivo y falso negativo que introdujimos anteriormente. Para completar el panorama, llamamos verdaderos positivos a las muestras correctamente clasificadas pertenecientes a la clase positiva, y verdaderos negativos a las muestras correctamente clasificadas pertenecientes a la clase negativa. Estos términos suelen abreviarse como FP, FN, TP y TN, y dan lugar a la siguiente interpretación de la matriz de confusión (Figura 5-11):

En[43]:

```
mglearn.plots.plot_binary_confusion_matrix()
```

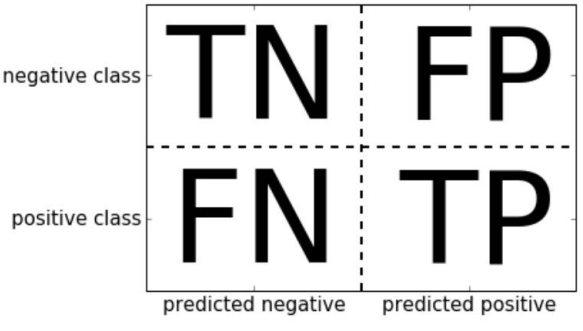

Figura 5-11. Matriz de confusión para la clasificación binaria

Ahora usemos la matriz de confusión para comparar los modelos que ajustamos anteriormente (los dos modelos ficticios, el árbol de decisión y la regresión logística):

En[44]:

```
print(" Clase más frecuente:")
print(matriz_confusión(prueba_y, pred_más_frecuente)) print("\ nModelo
ficticio:")
print(matriz_confusión(prueba_y, pred_ficticio)) print("\ nÁrbol
de decisión:")
print(matriz_confusión(prueba_y, pred_árbol)) print("\
nRegresión logística")
print(matriz_confusión(prueba_y, pred_logreg))
```

3 La diagonal principal de una matriz o conjunto bidimensional A es A[i, i].

Salida[44]:

```
Clase más frecuente: [[403
 0] [ 47 0]]

Modelo ficticio:
[[361 42]
  [ 43 4]]

Árbol de decisiones:

[[390 13] [ 24 23]]

Regresión logística [[401
 2] [ 8 39]]
```

Al observar la matriz de confusión, es evidente que algo falla con pred_most_frequent, ya que siempre predice la misma clase. Por otro lado, pred_dummy tiene un número muy bajo de verdaderos positivos (4), especialmente en comparación con el número de falsos negativos y falsos positivos: ¡hay muchos más falsos positivos que verdaderos positivos! Las predicciones del árbol de decisión son mucho más lógicas que las predicciones ficticias, aunque la precisión fue casi la misma.
Finalmente, podemos observar que la regresión logística es mejor que pred_tree en todos los aspectos: presenta más verdaderos positivos y verdaderos negativos, mientras que presenta menos falsos positivos y falsos negativos. De esta comparación, queda claro que solo el árbol de decisión y la regresión logística ofrecen resultados razonables, y que la regresión logística funciona mejor que el árbol en todos los aspectos. Sin embargo, inspeccionar la matriz de confusión completa es un poco engorroso, y si bien obtuvimos mucha información al analizar todos los aspectos de la matriz, el proceso fue muy manual y cualitativo. Hay varias maneras de resumir la información en la matriz de confusión, que analizaremos a continuación.

Relación con la precisión. Ya vimos una forma de resumir el resultado en la matriz de confusión: calculando la precisión, que se puede expresar como:

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

En otras palabras, la precisión es el número de predicciones correctas (TP y TN) dividido por el número de todas las muestras (todas las entradas de la matriz de confusión sumadas).

Precisión, recuperación y puntuación f. Existen otras maneras de resumir la matriz de confusión, siendo las más comunes la precisión y la recuperación. La precisión mide cuántas muestras predichas como positivas son realmente positivas:

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

La precisión se utiliza como métrica de rendimiento cuando el objetivo es limitar el número de falsos positivos. Por ejemplo, imaginemos un modelo para predecir la eficacia de un nuevo fármaco en el tratamiento de una enfermedad en ensayos clínicos. Los ensayos clínicos son notoriamente costosos, y una empresa farmacéutica solo querrá realizar un experimento si está completamente segura de que el fármaco realmente funcionará. Por lo tanto, es importante que el modelo no produzca muchos falsos positivos; en otras palabras, que tenga una alta precisión.
La precisión también se conoce como valor predictivo positivo (VPP).

La recuperación, por otro lado, mide cuántas de las muestras positivas son capturadas por las predicciones positivas:

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

La recuperación se utiliza como métrica de rendimiento cuando necesitamos identificar todas las muestras positivas; es decir, cuando es importante evitar falsos negativos. El ejemplo del diagnóstico de cáncer mencionado anteriormente en este capítulo es un buen ejemplo: es importante encontrar a todas las personas enfermas, posiblemente incluyendo a pacientes sanos en la predicción. La recuperación también se conoce como sensibilidad, tasa de aciertos o tasa de verdaderos positivos (TPR).

Existe un equilibrio entre optimizar la recuperación y optimizar la precisión. Se puede obtener una recuperación perfecta si se predice que todas las muestras pertenecen a la clase positiva; no habrá falsos negativos ni verdaderos negativos. Sin embargo, predecir todas las muestras como positivas resultará en muchos falsos positivos y, por lo tanto, la precisión será muy baja. Por otro lado, si se encuentra un modelo que predice solo el punto de datos que se considera más seguro como positivo y el resto como negativo, entonces la precisión será perfecta (suponiendo que este punto de datos sea, de hecho, positivo), pero la recuperación será muy baja.

La precisión y la recuperación son solo dos de las muchas medidas de clasificación derivadas de TP, FP, TN y FN. Puedes encontrar un excelente resumen de todas las medidas en Wikipedia. En la comunidad de aprendizaje automático, la precisión y la recuperación son posiblemente las medidas más utilizadas para la clasificación binaria, pero otras comunidades podrían utilizar otras métricas relacionadas.

Por lo tanto, si bien la precisión y la recuperación son medidas muy importantes, analizar solo una de ellas no ofrece una visión completa. Una forma de resumirlas es la puntuación f o medida f, que es la media armónica de la precisión y la recuperación:

$$F = 2 \cdot \frac{\text{precisión} \cdot \text{recuperación}}{\text{precisión} + \text{recuperación}}$$

Esta variante en particular también se conoce como puntuación f1 . Dado que considera la precisión y la recuperación, puede ser una mejor medida que la exactitud en conjuntos de datos de clasificación binaria desequilibrados. Ejecutémosla con las predicciones para el conjunto de datos "nueve vs. resto" que calculamos anteriormente. Aquí, asumiremos que la clase "nueve" es la clase positiva (se etiqueta como Verdadero mientras que el resto se etiqueta como Falso), por lo que la clase positiva es la clase minoritaria:

En[45]:

```
Desde sklearn.metrics, importar f1_score
print("Puntuación f1 más frecuente: {:.2f}".format( f1_score(y_test,
            pred_most_frequent))) print(" Puntuación f1 ficticia:
{:.2f}".format(f1_score(y_test, pred_dummy))) print("Puntuación f1 árbol:
{:.2f}".format(f1_score(y_test, pred_tree))) print("Puntuación f1 regresión logística:
{:.2f}".format( f1_score(y_test, pred_logreg)))
```

Salida[45]:

```
Puntuación f1 más frecuente: 0,00
Puntuación f1 ficticia: 0,10
Puntuación f1 árbol: 0,55
Puntuación f1 regresión logística: 0,89
```

Podemos observar dos cosas aquí. Primero, recibimos un mensaje de error para la predicción most_frequent , ya que no hubo predicciones de la clase positiva (lo que hace que el denominador en el f-score sea cero). Además, podemos ver una distinción bastante fuerte entre las predicciones ficticias y las predicciones del árbol, lo cual no era claro al observar solo la precisión. Usando el f-score para la evaluación, resumimos el rendimiento predictivo nuevamente en un número. Sin embargo, el f-score parece capturar nuestra intuición de lo que hace que un buen modelo sea mucho mejor que la precisión. Sin embargo, una desventaja del f-score es que es más difícil de interpretar y explicar que la precisión.

Si queremos un resumen más completo de la precisión, la recuperación y el puntaje f1 , podemos usar la función de conveniencia classification_report para calcular los tres a la vez e imprimirlos en un formato agradable:

En[46]:

```
de sklearn.metrics importar informe_de_clasificación
imprimir(informe_de_clasificación(prueba_y, pred_más_frecuente,
                                  nombres_objetivo=["no nueve", "nueve"]))
```

Salida[46]:

	precisión	recordar la compatibilidad con la puntuación f1		
no nueve	0.90	1.00	0,94	403
nueve	0.00	0.00	0,00	47
promedio/total	0.80	0.90	0,85	450

La función classification_report produce una línea por clase (aquí, True y Falso) e informa precisión, recuperación y puntuación f con esta clase como clase positiva. Antes, asumíamos que la clase minoritaria "nueve" era la clase positiva. Si cambiamos... clase positiva a “no nueve”, podemos verlo en la salida de classification_report que obtenemos una puntuación f de 0,94 con el modelo más frecuente . Además, para el En la clase “no nueve” tenemos un retiro de 1, ya que clasificamos todas las muestras como “no nueve”. La última columna junto al puntaje f proporciona el soporte de cada clase, lo que simplemente significa el número de muestras en esta clase según la verdad fundamental.

La última fila del informe de clasificación muestra una ponderación (por el número de muestras) en la clase) promedio de los números de cada clase. Aquí hay dos informes más, uno para el clasificador ficticio y uno para la regresión logística:

En[47]:

```
imprimir(informe_de_clasificación(prueba_y, pred_dummy,
                                  target_names=["no nueve", "nueve"]))
```

Salida[47]:

	precisión	recordar la compatibilidad con la puntuación f1		
no nueve	0,90	0,92	0,91	403
nueve	0,11	0,09	0,10	47
promedio/total	0.81	0.83	0,82	450

En[48]:

```
imprimir(informe_de_clasificación(prueba_y, pred_logreg,
                                  target_names=["no nueve", "nueve"]))
```

Salida[48]:

	precisión	recordar la compatibilidad con la puntuación f1		
no nueve	0,98	1.00	0,99	403
nueve	0,95	0.83	0,89	47
promedio/total	0,98	0,98	0,98	450

Como podrá observar al consultar los informes, las diferencias entre los modelos ficticios y un modelo muy bueno ya no son tan claras. Elegir la clase declarada positiva tiene un gran impacto en las métricas. Mientras que la puntuación f para la clasificación ficticia es de 0,13 (frente a 0,89 para la regresión logística) en la clase "nueve", para la clase "no nueve" es de 0,90 frente a 0,99, resultados que parecen razonables.
Sin embargo, si observamos todos los números juntos obtenemos un panorama bastante preciso y podemos ver claramente la superioridad del modelo de regresión logística.

Considerando la incertidumbre La

matriz de confusión y el informe de clasificación proporcionan un análisis muy detallado de un conjunto particular de predicciones. Sin embargo, las predicciones en sí mismas ya descartaron mucha información contenida en el modelo. Como discutimos en el Capítulo 2, la mayoría de los clasificadores proporcionan un método decision_function o predict_proba para evaluar los grados de certeza sobre las predicciones. Hacer predicciones puede verse como umbralizar la salida de decision_function o predict_proba en un punto fijo determinado: en la clasificación binaria usamos 0 para la función de decisión y 0,5 para predict_proba.

El siguiente es un ejemplo de una tarea de clasificación binaria desequilibrada, con 400 puntos de la clase negativa clasificados frente a 50 puntos de la positiva. Los datos de entrenamiento se muestran a la izquierda en la Figura 5-12. Entrenamos un modelo SVM de kernel con estos datos, y los gráficos a la derecha de los datos de entrenamiento ilustran los valores de la función de decisión como un mapa de calor. Se puede observar un círculo negro en el gráfico, en la parte superior central, que indica que el umbral de la función de decisión es exactamente cero. Los puntos dentro de este círculo se clasificarán como la clase positiva, y los puntos fuera de este círculo como la clase negativa.

En[49]:

```
de mglearn.datasets importar make_blobs X, y =
make_blobs(n_muestras=(400, 50), centros=2, cluster_std=[7.0, 2], estado_aleatorio=22)

X_entrenamiento, X_prueba, y_entrenamiento, y_prueba = tren_prueba_división(X, y, estado_aleatorio=0)
svc = SVC(gamma=.05).fit(X_entrenamiento, y_entrenamiento)
```

En[50]:

```
mglearn.plots.umbral_de_decisión_del_plot()
```

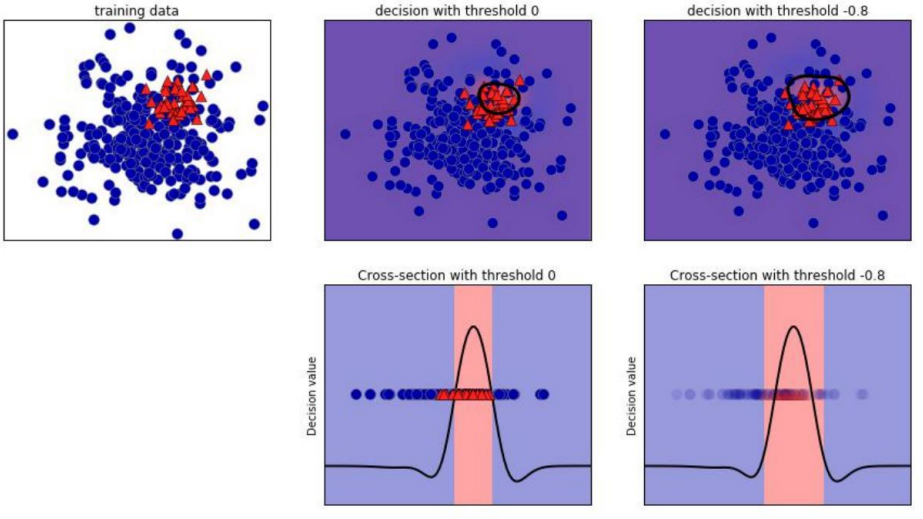

Figura 5-12. Mapa de calor de la función de decisión y el impacto de cambiar la decisión.
límite

Podemos utilizar la función classification_report para evaluar la precisión y la recuperación de
ambas clases:

En[51]:

```
imprimir(informe_de_clasificación(prueba_y, svc.predict(prueba_X)))
```

Salida[51]:

	precisión	recordar la compatibilidad con la puntuación f1		
0	0,97	0.89	0,93	104
1	0.35	0.67	0.46	9
promedio/total	0,92	0.88	0.89	113

Para la clase 1, obtenemos una recuperación bastante baja y la precisión es mixta. Debido a que la clase 0 es tan...
mucho más grande, el clasificador se centra en obtener correctamente la clase 0, y no la clase 1, más pequeña.

Supongamos que en nuestra aplicación es más importante tener una alta recuperación para la clase 1, ya que
En el ejemplo anterior de detección del cáncer, esto significa que estamos dispuestos a arriesgarnos a más resultados falsos.
positivos (falsos de clase 1) a cambio de más verdaderos positivos (que aumentarán la
recordar). Las predicciones generadas por svc.predict realmente no cumplen con este requisito.
mento, pero podemos ajustar las predicciones para centrarnos en un mayor recuerdo de la clase 1 mediante
cambiando el umbral de decisión de 0. De forma predeterminada, los puntos con un umbral de decisión
Un valor de sion_function mayor que 0 se clasificará como clase 1. Queremos más puntos.
para ser clasificado como clase 1, por lo que necesitamos disminuir el umbral:

En[52]:

```
y_pred_umbral_inferior = svc.función_de_decisión(X_test) > -.8
```

Veamos el informe de clasificación para esta predicción:

En[53]:

```
imprimir(informe_de_clasificación(prueba_y, umbral_inferior_y_pred))
```

Salida[53]:

	precisión	recordar la compatibilidad con la puntuación f1		
0	1.00	0,82	0.90	104
1	0.32	1.00	0.49	9
promedio/total	0,95	0.83	0.87	113

Como era de esperar, la recuperación de la clase 1 aumentó y la precisión disminuyó. Ahora estamos...
clasificar una región más grande del espacio como clase 1, como se ilustra en el panel superior derecho de
Figura 5-12. Si valora la precisión sobre la recuperación o viceversa, o sus datos son
Si la situación es muy desequilibrada, cambiar el umbral de decisión es la forma más fácil de obtener una apuesta.
resultados. Como la función de decisión puede tener rangos arbitrarios, es difícil proporcionar
Una regla general sobre cómo elegir un umbral.

Si establece un umbral, debe tener cuidado de no hacerlo utilizando
el conjunto de prueba. Al igual que con cualquier otro parámetro, establecer un umbral de decisión...
Es probable que la información antigua del conjunto de prueba arroje resultados demasiado optimistas. Utilice un
conjunto de validación o validación cruzada en su lugar.

La elección de un umbral para los modelos que implementan el método predict_proba puede ser
más fácil, ya que la salida de predict_proba está en una escala fija de 0 a 1 y modela la probabilidad.
idades. Por defecto, el umbral de 0,5 significa que si el modelo tiene más del 50 % de seguridad
Si un punto es de clase positiva, se clasificará como tal. Aumentar el umbral...
Viejo significa que el modelo necesita tener más confianza para tomar una decisión positiva.
(y menos seguros para tomar una decisión negativa). Al trabajar con probabilidades
Puede ser más intuitivo que trabajar con umbrales arbitrarios, pero no todos los modelos proporcionan
modelos realistas de incertidumbre (un DecisionTree desarrollado hasta su máxima profundidad)
siempre 100% seguro de sus decisiones, aunque a menudo pueda equivocarse). Esto se relaciona
al concepto de calibración: un modelo calibrado es un modelo que proporciona una
Medida de su incertidumbre. Discutir la calibración en detalle está fuera del alcance de este
libro, pero puedes encontrar más detalles en el artículo "Predicción de buenas probabilidades con
Aprendizaje supervisado" de Alexandru Niculescu-Mizil y Rich Caruana.

Como acabamos de discutir, cambiar el umbral que se utiliza para tomar una decisión de clasificación en un modelo es una forma de ajustar el equilibrio entre precisión y recuperación para un clasificador determinado. Tal vez desee omitir menos del 10% de las muestras positivas, lo que significa una recuperación deseada del 90%. Esta decisión depende de la aplicación y debe estar impulsada por los objetivos comerciales. Una vez que se establece un objetivo particular, por ejemplo, un valor particular de recuperación o precisión para una clase, se puede establecer un umbral adecuadamente. Siempre es posible establecer un umbral para cumplir un objetivo particular, como el 90% de recuperación. La parte difícil es desarrollar un modelo que aún tenga una precisión razonable con este umbral: si clasifica todo como positivo, tendrá una recuperación del 100%, pero su modelo será inútil.

Establecer un requisito en un clasificador, como un 90% de recuperación, suele denominarse establecer el punto de operación. Fijar un punto de operación suele ser útil en entornos empresariales para garantizar el rendimiento a los clientes u otros grupos de la organización.

A menudo, al desarrollar un nuevo modelo, no está del todo claro cuál será el punto de operación. Por esta razón, y para comprender mejor un problema de modelado, resulta instructivo examinar todos los umbrales posibles, o todas las posibles compensaciones entre precisión y recuperación, a la vez. Esto es posible mediante una herramienta llamada curva de precisión-recuperación. Puede encontrar la función para calcular la curva de precisión-recuperación en el módulo sklearn.metrics . Necesita el etiquetado de la verdad fundamental y las incertidumbres predichas, creadas mediante decision_function o predict_proba:

En[54]:

```
de sklearn.metrics importar precision_recall_curve precisión,
recuperación, umbrales = precision_recall_curve( y_test,
      svc.decision_function(X_test))
```

La función precision_recall_curve devuelve una lista de valores de precisión y recuperación para todos los umbrales posibles (todos los valores que aparecen en la función de decisión) en orden ordenado, de modo que podemos trazar una curva, como se ve en la Figura 5-13:

En[55]:

```
# Utilice más puntos de datos para una curva más
suave X, y = make_blobs(n_samples=(4000, 500), centers=2, cluster_std=[7.0, 2],
                        random_state=22)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0) svc =
SVC(gamma=.05).fit(X_train, y_train) precisión,
recall, umbrales = precision_recall_curve( y_test,
      svc.decision_function(X_test)) # encontrar el
umbral más cercano a cero close_zero
= np.argmin(np.abs(thresholds))
plt.plot(precision[close_zero], recall[close_zero], 'o', markersize=10, label="umbral cero",
            fillstyle="none", c='k', mew=2)

plt.plot(precision, recall, label="curva de precisión y recall")
plt.xlabel("Precisión")
plt.ylabel("Recall")
```

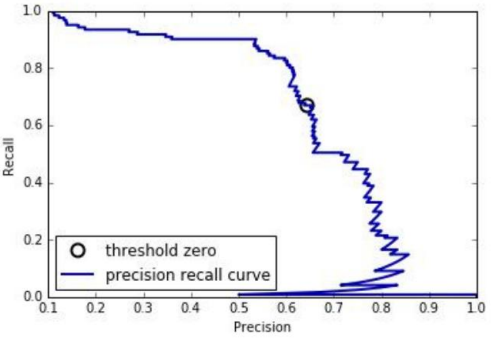

Figura 5-13. Curva de recuperación de precisión para SVC (gamma = 0,05)

Cada punto a lo largo de la curva de la Figura 5-13 corresponde a un posible umbral de la función de decisión. Podemos observar, por ejemplo, que podemos lograr una recuperación de 0,4 con una precisión de aproximadamente 0,75. El círculo negro marca el punto que corresponde a un umbral de 0, el umbral predeterminado para la función de decisión. Este punto es el equilibrio que se elige al llamar al método de predicción .

Cuanto más cerca se mantenga una curva de la esquina superior derecha, mejor será el clasificador. Un punto en la esquina superior derecha significa alta precisión y alta recuperación para el mismo umbral. La curva comienza en la esquina superior izquierda, lo que corresponde a un umbral muy bajo, clasificando todo como la clase positiva. Al aumentar el umbral, la curva avanza hacia una mayor precisión, pero también hacia una menor recuperación. Al aumentar el umbral cada vez más, llegamos a una situación en la que la mayoría de los puntos clasificados como positivos son verdaderos positivos, lo que resulta en una precisión muy alta, pero una menor recuperación. Cuanto más alto mantenga el modelo la recuperación a medida que aumenta la precisión, mejor.

Al observar esta curva en particular con más detalle, podemos ver que con este modelo es posible obtener una precisión de hasta aproximadamente 0,5 con una recuperación muy alta. Si deseamos una precisión mucho mayor, debemos sacrificar mucha recuperación. En otras palabras, a la izquierda la curva es relativamente plana, lo que significa que la recuperación no disminuye mucho cuando se requiere mayor precisión. Para una precisión superior a 0,5, cada aumento de precisión nos cuesta mucha recuperación.

Diferentes clasificadores pueden funcionar correctamente en diferentes partes de la curva, es decir, en diferentes puntos operativos. Comparemos el SVM que entrenamos con un bosque aleatorio entrenado con el mismo conjunto de datos. El RandomForestClassifier no tiene una función de decisión, solo predict_proba. La función precision_recall_curve espera como segundo argumento una medida de certeza para la clase positiva (clase 1), por lo que pasamos la probabilidad de que una muestra sea de clase 1, es decir, rf.predict_proba(X_test)[:, 1]. El umbral predeterminado para predict_proba en la clasificación binaria es 0,5, por lo que este es el punto que marcamos en la curva (véase la Figura 5-14).

En[56]:

```
de sklearn.ensemble importar RandomForestClassifier

rf = RandomForestClassifier(n_estimadores=100, estado_aleatorio=0, características_máximas=2)
rf.fit(X_train, y_train)

# RandomForestClassifier tiene predict_proba, pero no decision_function precision_rf,
recall_rf, thresholds_rf = precision_recall_curve(
      prueba_y, rf.predict_proba(prueba_X)[:, 1])

plt.plot(precisión, recuperación, etiqueta="svc")

plt.plot(precisión[cerrar_cero], recuperación[cerrar_cero], 'o', tamaño del marcador=10,
            etiqueta="umbral cero svc", estilo de relleno="ninguno", c='k', mew=2)

plt.plot(precisión_rf, recuperación_rf, etiqueta="rf")

cierre_default_rf = np.argmin(np.abs(umbrales_de_rf - 0.5))
plt.plot(precisión_rf[cierre_default_rf], recuperación_rf[cierre_default_rf], '^', c='k',
            tamaño_de_marcador=10, etiqueta="umbral 0.5_rf", estilo_de_relleno="ninguno", mew=2)
plt.xlabel("Precisión")
plt.ylabel("Recordar")
plt.legend(loc="mejor")
```

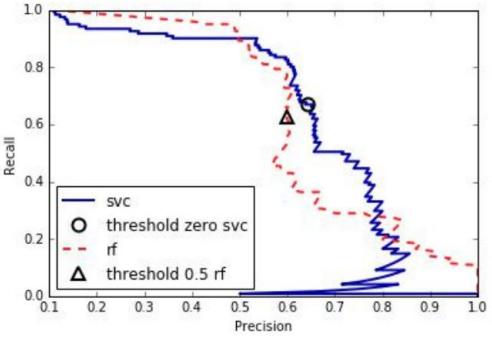

Figura 5-14. Comparación de las curvas de recuperación de precisión de SVM y bosque aleatorio

Del gráfico de comparación se desprende que el bosque aleatorio ofrece un mejor rendimiento en los extremos, con requisitos de recuperación o precisión muy altos. En el punto medio (precisión aproximada de 0,7), el SVM ofrece un mejor rendimiento. Si solo hubiéramos analizado la puntuación f1 para comparar el rendimiento general, habríamos pasado por alto estas sutilezas. La puntuación f1 solo captura un punto en la curva de precisión-recuperación, el dado por el umbral predeterminado:

En[57]:

```
print("puntuación f1 del bosque aleatorio: {:.3f}".format( puntuación
      f1(prueba_y, rf.predict(prueba_X))))
print("f1_score de svc: {:.3f}".format(f1_score(y_test, svc.predict(X_test))))
```

Salida[57]:

```
Puntuación f1 del bosque aleatorio: 0,610
Puntuación f1 del svc: 0,656
```

Comparar dos curvas de precisión-recuperación proporciona información detallada, pero es un proceso bastante manual. Para una comparación automática de modelos, podríamos resumir la información de la curva, sin limitarnos a un umbral o punto de operación específico. Una forma particular de resumir la curva de precisión-recuperación es calculando la integral o área bajo la curva de la misma, también conocida como precisión promedio.
[4] Puede usar la función average_precision_score para calcular la precisión promedio. Dado que necesitamos calcular la curva ROC y considerar múltiples umbrales, el resultado de decision_function o predict_proba debe pasarse a average_precision_score, no el resultado de predict:

En[58]:

```
de sklearn.metrics importar puntuación_de_precisión_promedio
ap_rf = puntuación_de_precisión_promedio(prueba_y, rf.predict_proba(prueba_X)[:, 1])
ap_svc = puntuación_de_precisión_promedio(prueba_y, svc.función_de_decisión(prueba_X))
imprimir("Precisión promedio del bosque aleatorio: {:.3f}".format(ap_rf))
imprimir("Precisión promedio de svc: {:.3f}".format(ap_svc))
```

Salida[58]:

```
Precisión media del bosque aleatorio: 0,666 Precisión
media de svc: 0,663
```

Al promediar todos los umbrales posibles, observamos que el bosque aleatorio y el SVC tienen un rendimiento similar, con el bosque aleatorio incluso ligeramente por delante. Esto difiere bastante del resultado obtenido anteriormente con f1_score . Dado que la precisión promedio es el área bajo una curva que va de 0 a 1, siempre devuelve un valor entre 0 (peor) y 1 (mejor). La precisión promedio de un clasificador que asigna la función de decisión aleatoriamente es la fracción de muestras positivas en el conjunto de datos.

Características operativas del receptor (ROC) y

AUC. Existe otra herramienta que se utiliza comúnmente para analizar el comportamiento de los clasificadores en diferentes umbrales: la curva de características operativas del receptor, o curva ROC. Similar a la curva de precisión-recuperación, la curva ROC considera todos los posibles...

---

4 Existen algunas diferencias técnicas menores entre el área bajo la curva de precisión-recuperación y el promedio precisión. Sin embargo, esta explicación transmite la idea general.

Umbrales para un clasificador dado, pero en lugar de reportar la precisión y la recuperación, muestra la tasa de falsos positivos (FPR) en comparación con la tasa de verdaderos positivos (TPR). Recuerde que la tasa de verdaderos positivos es simplemente otro nombre para la recuperación, mientras que la tasa de falsos positivos es la fracción de falsos positivos de todas las muestras negativas.

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

La curva ROC se puede calcular utilizando la función roc_curve (ver Figura 5-15):

En[59]:

```
desde sklearn.metrics importar roc_curve fpr,
tpr, umbrales = roc_curve(y_test, svc.decision_function(X_test))

plt.plot(fpr, tpr, label=" Curva ROC")
plt.xlabel("FPR")
plt.ylabel("TPR (recordar)") #
encontrar el umbral más cercano a cero
close_zero = np.argmin(np.abs(umbrales))
plt.plot(fpr[close_zero], tpr[close_zero], 'o', markersize=10,
         etiqueta="umbral cero", estilo de relleno="ninguno", c='k', mew=2)
plt.legend(loc=4)
```

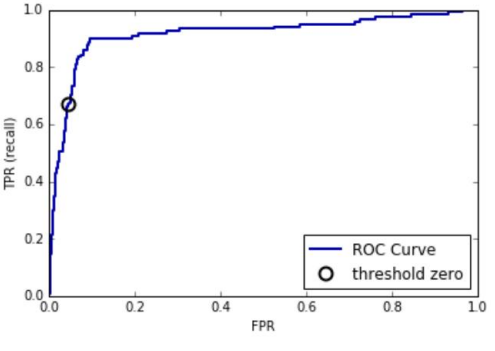

Figura 5-15. Curva ROC para SVM

Para la curva ROC, la curva ideal se encuentra cerca de la esquina superior izquierda: se busca un clasificador que produzca una alta tasa de recuperación y mantenga una baja tasa de falsos positivos. En comparación con el umbral predeterminado de 0, la curva muestra que podemos lograr una tasa de recuperación significativamente mayor (alrededor de 0,9) con un ligero aumento del FPR. El punto más cercano a la esquina superior izquierda podría ser un mejor punto de operación que el elegido por defecto. Cabe mencionar que la elección de un umbral no debe realizarse en el conjunto de prueba, sino en una validación independiente.

colocar.

Puede encontrar una comparación del bosque aleatorio y el SVM utilizando curvas ROC en Figura 5-16:

En[60]:

```
de sklearn.metrics importar roc_curve fpr_rf,
tpr_rf, umbrales_rf = roc_curve(y_test, rf.predict_proba(X_test)[:, 1])

plt.plot(fpr, tpr, etiqueta=" Curva ROC SVC")
plt.plot(fpr_rf, tpr_rf, etiqueta=" Curva ROC RF")

plt.xlabel("FPR")
plt.ylabel("TPR (recuperación)")
plt.plot(fpr[cerrar_cero], tpr[cerrar_cero], 'o', tamaño_marcador=10,
         etiqueta="umbral cero SVC", estilo de relleno="ninguno", c='k', mew=2)
cierre_rf_predeterminado = np.argmin(np.abs(umbrales_rf - 0.5))
plt.plot(fpr_rf[cierre_rf_predeterminado], tpr[cierre_rf_predeterminado], '^', tamaño_marcador=10,
         etiqueta="umbral 0.5 RF", estilo_de_relleno="ninguno", c='k', mew=2)

plt.legend(loc=4)
```

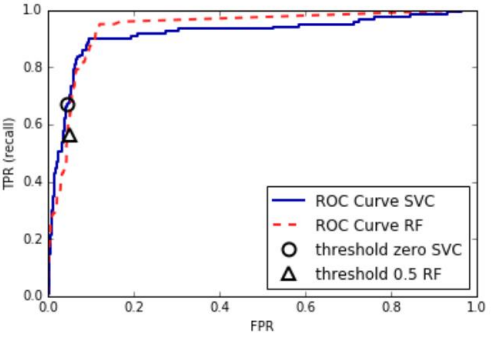

Figura 5-16. Comparación de las curvas ROC para SVM y bosque aleatorio

En cuanto a la curva de precisión-recuperación, a menudo queremos resumir la curva ROC mediante un único número: el área bajo la curva (comúnmente denominada AUC, y se entiende que la curva en cuestión es la curva ROC). Podemos calcular el área bajo la curva ROC mediante la función roc_auc_score :

En[61]:

```
de sklearn.metrics importar roc_auc_score rf_auc =
roc_auc_score(y_test, rf.predict_proba(X_test)[:, 1]) svc_auc = roc_auc_score(y_test,
svc.decision_function(X_test)) imprimir("AUC para Bosque aleatorio: {:.3f}".format(rf_auc))
imprimir("AUC para SVC: {:.3f}".format(svc_auc))
```

Salida[61]:

```
AUC para Bosque aleatorio: 0,937
AUC para SVC: 0,916
```

Comparando el bosque aleatorio y el SVM usando la puntuación AUC, encontramos que el bosque aleatorio funciona bastante mejor que el SVM. Recordemos que debido a que la precisión promedio es el área bajo una curva que va de 0 a 1, la precisión promedio siempre devuelve un valor entre 0 (peor) y 1 (mejor). Predecir aleatoriamente siempre produce un AUC de 0,5, sin importar cuán desequilibradas estén las clases en un conjunto de datos. Esto hace que el AUC sea una métrica mucho mejor para problemas de clasificación desequilibrada que la precisión. El AUC puede interpretarse como una evaluación de la clasificación de muestras positivas. Es equivalente a la probabilidad de que un punto elegido aleatoriamente de la clase positiva tenga una puntuación más alta según el clasificador que un punto elegido aleatoriamente de la clase negativa. Entonces, un AUC perfecto de 1 significa que todos los puntos positivos tienen una puntuación más alta que todos los puntos negativos. Para problemas de clasificación con clases desequilibradas, usar el AUC para la selección del modelo suele ser mucho más significativo que usar la precisión.

Retomemos el problema que estudiamos anteriormente: clasificar todos los nueves en el conjunto de datos de dígitos frente a los demás dígitos. Clasificaremos el conjunto de datos con una máquina de modelado de números (SVM) con tres configuraciones diferentes del ancho de banda del kernel, gamma (véase la Figura 5-17):

En[62]:

```
y = dígitos.objetivo == 9

X_train, X_test, y_train, y_test = train_test_split( dígitos.datos, y,
        estado_aleatorio=0)

plt.figura()

Para gamma en [1, 0.05, 0.01]: svc =
        SVC(gamma=gamma).fit(X_train, y_train) precisión =
        svc.score(X_test, y_test) auc = roc_auc_score(y_test,
        svc.decision_function(X_test)) fpr, tpr, _ = roc_curve(y_test , svc.decision_function(X_test))
        print("gamma = {:.2f} precisión = {:.2f} AUC = {:.2f}".format( gamma, precisión, auc)) plt.plot(fpr,
        tpr, label="gamma={:.3f}".format(gamma)) plt.xlabel("FPR") plt.ylabel("TPR") plt.xlim(-0.01, 1)
        plt.ylim(0, 1.02)
        plt.legend(loc="mejor")
```

Salida[62]:

```
gamma = 1,00 precisión = 0,90 AUC = 0,50 gamma =
0,05 precisión = 0,90 AUC = 0,90 gamma = 0,01
precisión = 0,90 AUC = 1,00
```

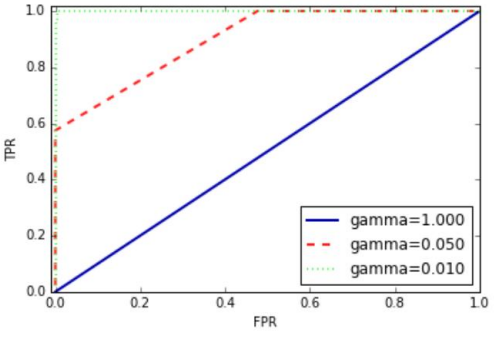

Figura 5-17. Comparación de las curvas ROC de las SVM con diferentes configuraciones de gamma.

La precisión de los tres ajustes de gamma es la misma, 90%. Esto podría ser lo mismo que el rendimiento aleatorio, o podría no serlo. Sin embargo, al observar el AUC y la curva correspondiente, vemos una clara distinción entre los tres modelos. Con gamma = 1.0, el AUC está en realidad al nivel de azar, lo que significa que la salida de la función de decisión es tan buena como aleatoria. Con gamma = 0.05, el rendimiento mejora drásticamente a un AUC de 0.5. Finalmente, con gamma = 0.01, obtenemos un AUC perfecto de 1.0. Eso significa que todos los puntos positivos se clasifican por encima de todos los puntos negativos según la función de decisión. En otras palabras, con el umbral correcto, ¡este modelo puede clasificar los datos perfectamente![5] Sabiendo esto, podemos ajustar el umbral en este modelo y obtener grandes predicciones. Si solo hubiéramos usado la precisión, nunca habríamos descubierto esto.

Por esta razón, recomendamos encarecidamente usar el AUC al evaluar modelos con datos desequilibrados. Tenga en cuenta que el AUC no utiliza el umbral predeterminado, por lo que podría ser necesario ajustar el umbral de decisión para obtener resultados de clasificación útiles de un modelo con un AUC alto.

Métricas para la clasificación multiclase

Ahora que hemos analizado a fondo la evaluación de las tareas de clasificación binaria, pasemos a las métricas para evaluar la clasificación multiclase. Básicamente, todas las métricas para la clasificación multiclase se derivan de las métricas de clasificación binaria, pero se promedian.

---

[5] Al observar la curva de gamma=0,01 en detalle, se puede observar una pequeña curva cerca de la esquina superior izquierda. Esto significa que al menos un punto no se clasificó correctamente. El AUC de 1,0 se debe al redondeo al segundo decimal.

En todas las clases. La precisión para la clasificación multiclase se define nuevamente como la fracción de ejemplos correctamente clasificados. Y, de nuevo, cuando las clases están desequilibradas, la precisión no es una buena medida de evaluación. Imaginemos un problema de clasificación de tres clases con el 85 % de los puntos correspondientes a la clase A, el 10 % a la clase B y el 5 % a la clase C. ¿Qué significa tener una precisión del 85 % en este conjunto de datos? En general, los resultados de la clasificación multiclase son más difíciles de comprender que los de la clasificación binaria. Además de la precisión, las herramientas comunes son la matriz de confusión y el informe de clasificación que vimos en el caso binario en la sección anterior. Apliquemos estos dos métodos de evaluación detallados a la tarea de clasificar los 10 dígitos manuscritos diferentes en el conjunto de datos de dígitos :

En[63]:

```
de sklearn.metrics importar puntuación_de_precisión X_train,
X_test, y_train, y_test = train_test_split( dígitos.datos, dígitos.objetivo,
        estado_aleatorio=0) lr = LogisticRegression().fit(X_train, y_train)
pred = lr.predict(X_test) imprimir("Precisión:

{:.3f}".format(puntuación_de_precisión(y_test, pred))) imprimir(" Matriz de confusión:
\n{}".format(matriz_de_confusión(y_test, pred)))
```

Salida[63]:

```
Precisión: 0,953 Matriz
de confusión: [[37 0 0 0
 0 0 0 0 0 0] [ 0 39 0 0 0 0 2 0 2 0] [ 0 0 41 3 0 0
  0 0 0 0] [ 0 0 1 43 0 0 0 0 0 1] [ 0 0 0 0 38 0 0
  0 0 0] [ 0 1 0 0 0 47 0 0 0 0] [ 0 0 0 0 0 0 52 0
  0 0] [ 0 1 0 1 1 0 0 45 0 0] [ 0 3 1 0 0 0 0 0 43
  1] [ 0 0 0 1 0 1 0 0 1 44]]
```

El modelo tiene una precisión del 95,3%, lo que ya nos indica que lo estamos haciendo bastante bien. La matriz de confusión nos proporciona más detalles. En el caso binario, cada fila corresponde a una etiqueta verdadera y cada columna a una etiqueta predicha. Puede encontrar un gráfico visualmente más atractivo en la Figura 5-18:

En[64]:

```
puntuaciones_imagen =
        mglearn.tools.heatmap( matriz_confusión(y_test, pred), xlabel='Etiqueta predicha',
        ylabel=' Etiqueta verdadera ', xticklabels=dígitos.nombres_objetivo,
        yticklabels=dígitos.nombres_objetivo, cmap=plt.cm.gray_r, fmt="%d")
plt.title("Matriz de confusión")
plt.gca().invert_yaxis()
```

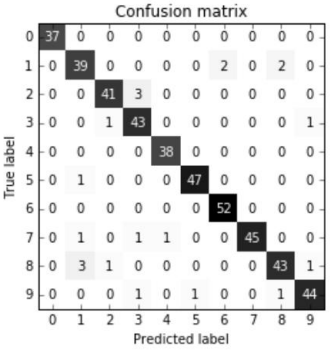

Figura 5-18. Matriz de confusión para la tarea de clasificación de 10 dígitos

Para la primera clase, el dígito 0, hay 37 muestras en la clase, y todas estas son iguales.
Los ejemplos se clasificaron como clase 0 (no hay falsos negativos para la clase 0). Podemos ver que
porque todas las demás entradas en la primera fila de la matriz de confusión son 0. También podemos ver
que ningún otro dígito fue clasificado erróneamente como 0, porque todas las demás entradas en el
La primera columna de la matriz de confusión es 0 (no hay falsos positivos para la clase 0).
Sin embargo, algunos dígitos se confundían con otros, por ejemplo, el dígito 2 (tercera fila),
Tres de los cuales se clasificaron como el dígito 3 (cuarta columna). También había un dígito
3 que se clasificó como 2 (tercera columna, cuarta fila) y un dígito 8 que se clasificó
como 2 (tercera columna, cuarta fila).

Con la función classification_report , podemos calcular la precisión, la recuperación,
y puntuación f para cada clase:

En[65]:

```
imprimir(informe_de_clasificación(prueba_y, pred))
```

Salida[65]:

	precisión	recordar la compatibilidad con la puntuación f1		
0	1.00	1.00	1.00	37
1	0.89	0,91	0.90	43
2	0,95	0,93	0,94	44
	0,90	0,96	0,92	45
3 4	0,97	1,00	0,99	38
5	0,98	0,98	0,98	48
6	0,96	1.00	0,98	52
7	1.00	0,94	0,97	48
8	0,93	0.90	0,91	48
9	0,96	0.94	0,95	47
promedio/total	0,95	0,95	0,95	450

Como era de esperar, la precisión y la recuperación son un 1 perfecto para la clase 0, ya que no hay confusiones con esta clase. Para la clase 7, por otro lado, la precisión es 1 porque ninguna otra clase se clasificó erróneamente como 7, mientras que para la clase 6, no hay falsos negativos, por lo que la recuperación es 1. También podemos observar que el modelo tiene dificultades particulares con las clases 8 y 3.

La métrica más utilizada para conjuntos de datos desequilibrados en el entorno multiclase es la versión multiclase de la puntuación f. La idea de la puntuación f multiclase es calcular una puntuación f binaria por clase, donde dicha clase es la positiva y las demás, las negativas. Posteriormente, estas puntuaciones f por clase se promedian mediante una de las siguientes estrategias:

- El promedio "macro" calcula las puntuaciones f no ponderadas por clase. Esto da como resultado... peso para todas las clases, sin importar su tamaño.
- El promedio "ponderado" calcula la media de los puntajes f por clase, ponderados por Su apoyo. Esto es lo que consta en el informe de clasificación.
- El promedio "micro" calcula el número total de falsos positivos, falsos negativos y verdaderos positivos en todas las clases y luego calcula la precisión, la recuperación y el puntaje f utilizando estos recuentos.

Si le importan por igual todas las muestras, se recomienda usar el puntaje f1 promedio "micro" ; si le importan por igual todas las clases, se recomienda usar el puntaje f1 promedio "macro" .

En[66]:

```
print("Puntaje f1 promedio micro: {:.3f}".format
      (f1_score(y_test, pred, promedio="micro")))
print("Puntaje promedio f1 del macro: {:.3f}".format
      (f1_score(y_test, pred, promedio="macro")))
```

Salida[66]:

```
Puntuación f1 media micro: 0,953
Puntuación f1 media macro: 0,954
```

La evaluación de métricas

de regresión se puede realizar con un nivel de detalle similar al que realizamos para la clasificación; por ejemplo, analizando la predicción excesiva del objetivo frente a la predicción insuficiente del mismo. Sin embargo, en la mayoría de las aplicaciones que hemos visto, basta con usar el R predeterminado del método de puntuación de todos los regresores. En ocasiones, las decisiones empresariales se toman con base en el error cuadrático medio o el error absoluto medio, lo que podría incentivar el ajuste de los modelos utilizando estas métricas. En general, sin embargo, hemos encontrado que R es una métrica más intuitiva para evaluar modelos de regresión.

Uso de métricas de evaluación en la selección de modelos.

Hemos analizado en detalle muchos métodos de evaluación y cómo aplicarlos dados los datos reales y un modelo. Sin embargo, a menudo deseamos usar métricas como el AUC en la selección de modelos mediante GridSearchCV o cross_val_score. Afortunadamente, scikit-learn ofrece una forma muy sencilla de lograrlo mediante el argumento de puntuación , que puede usarse tanto en GridSearchCV como en cross_val_score. Simplemente se puede proporcionar una cadena que describa la métrica de evaluación que se desea utilizar. Por ejemplo, supongamos que queremos evaluar el clasificador SVM en la tarea "nueve vs. resto" del conjunto de datos de dígitos , utilizando la puntuación del AUC. Se puede cambiar la puntuación predeterminada (precisión) al AUC proporcionando "roc_auc" como parámetro de puntuación .

En[67]:

```
# La puntuación predeterminada para la clasificación es la
precisión print("Puntuación predeterminada:
      {}".format( cross_val_score(SVC(), digits.data, digits.target == 9)))
# proporcionar puntuación="precisión" no cambia los resultados
explicit_accuracy = cross_val_score(SVC(), digits.data, digits.target == 9, puntuación="precisión")

print("Puntuación de precisión explícita: {}".format(explicit_accuracy)) roc_auc =
cross_val_score(SVC(), digits.data, digits.target == 9,
                                        puntuación="roc_auc")
print(" Puntuación AUC: {}".format(roc_auc))
```

Salida[67]:

```
Puntuación predeterminada: [ 0.9 0.9 0.9]
Puntuación de precisión explícita: [0,9 0,9 0,9]
Puntuación AUC: [ 0,994 0,99 0,996]
```

De manera similar, podemos cambiar la métrica utilizada para elegir los mejores parámetros en Grid. BuscarCV:

En[68]:

```
X_train, X_test, y_train, y_test = train_test_split( dígitos.datos,
      dígitos.objetivo == 9, estado_aleatorio=0)

# proporcionamos una cuadrícula algo mala para ilustrar el punto:
param_grid = {'gamma': [0.0001, 0.01, 0.1, 1, 10]} # usando la
puntuación predeterminada de precisión: grid =
GridSearchCV(SVC(), param_grid=param_grid) grid.fit(X_train,
y_train) print("Búsqueda en
cuadrícula con precisión") print("Mejores
parámetros:", grid.best_params_) print("Mejor
puntuación de validación cruzada (precisión)): {:.3f}".format(grid.best_score_)) print(" AUC del conjunto
de pruebas: {:.3f}".format( roc_auc_score(y_test,
      grid.decision_function(X_test)))) print(" Precisión del conjunto de
pruebas: {:.3f}".format(grid.score(X_test, y_test)))
```

Salida[68]:

```
Búsqueda en cuadrícula con precisión
Mejores parámetros: {'gamma': 0.0001}
Mejor puntuación de validación cruzada (precisión)): 0,970 AUC del conjunto
de prueba: 0,992
Precisión del conjunto de prueba: 0,973
```

En[69]:

```
# usando la puntuación AUC en su lugar: grid
= GridSearchCV(SVC(), param_grid=param_grid, scoring="roc_auc") grid.fit(X_train, y_train) print("\nBúsqueda en
cuadrícula con AUC") print("Mejores
parámetros:", grid.best_params_) print("Mejor
puntuación de validación cruzada (AUC): {:.3f}".format(grid.best_score_))
print(" AUC del conjunto de pruebas: {:.3f}".format( roc_auc_score(y_test, grid.decision_function(X_test)))) print(" Precisión del
conjunto de pruebas: {:.3f}".format(grid.score(X_test, y_test)))
```

Salida[69]:

```
Búsqueda en cuadrícula con AUC
Mejores parámetros: {'gamma': 0.01}
Mejor puntuación de validación cruzada (AUC): 0,997 AUC del
conjunto de prueba: 1,000
Precisión del conjunto de prueba: 1.000
```

Al usar la precisión, se selecciona el parámetro gamma = 0,0001 , mientras que al usar el AUC se selecciona gamma = 0,01 . La precisión de la validación cruzada es consistente con la precisión del conjunto de prueba en ambos casos. Sin embargo, al usar el AUC se encontró una mejor configuración del parámetro en términos de AUC e incluso en términos de precisión.

Los valores más importantes del parámetro de puntuación para la clasificación son la precisión (predeterminado); roc_auc para el área bajo la curva ROC; average_precision para el área bajo la curva de precisión-recuperación; f1, f1_macro, f1_micro y f1_weighted para la puntuación binaria f1 y sus diferentes variantes ponderadas. Para la regresión, los valores más comunes son r^2 para la puntuación R, mean_squared_error para el error cuadrático medio y mean_absolute_error para el error absoluto medio. Puede encontrar una lista completa de los argumentos admitidos en la documentación. o mirando el diccionario SCORER definido en el módulo metrics.scorer :

6 Encontrar una solución de mayor precisión usando AUC es probablemente una consecuencia de que la precisión es una mala medida de Rendimiento del modelo en datos desequilibrados.

En[70]:

```
de sklearn.metrics.scorer importar SCORERS print("Calificadores
disponibles :\n{}".format(sorted(SCORERS.keys())))
```

Salida[70]:

```
Puntuadores disponibles:
['accuracy', 'adjusted_rand_score', 'average_precision', 'f1', 'f1_macro',
 'f1_micro', 'f1_muestras', 'f1_ponderado', 'pérdida logarítmica', 'error absoluto medio', 'error cuadrático medio',
 'error absoluto mediano', 'precisión', 'precisión_macro', 'precisión_micro', 'muestras_precisión', 'ponderado_precisión',
 'r2', 'recuperación', 'recuperación_macro', 'recuperación_micro', 'muestras_recuperación', 'ponderado_recuperación',
 'roc_auc']
```

Resumen y perspectivas

En este capítulo, analizamos la validación cruzada, la búsqueda en cuadrícula y las métricas de evaluación, pilares para evaluar y mejorar los algoritmos de aprendizaje automático. Las herramientas descritas en este capítulo, junto con los algoritmos descritos en los capítulos 2 y 3, son fundamentales para cualquier profesional del aprendizaje automático.

Hay dos puntos particulares que abordamos en este capítulo que merecen ser repetidos, ya que los nuevos profesionales a menudo los pasan por alto. El primero tiene que ver con la validación cruzada. La validación cruzada o el uso de un conjunto de pruebas nos permite evaluar un modelo de aprendizaje automático tal como se desempeñará en el futuro. Sin embargo, si usamos el conjunto de pruebas o la validación cruzada para seleccionar un modelo o seleccionar parámetros del modelo, "agotamos" los datos de prueba, y usar los mismos datos para evaluar qué tan bien se desempeñará nuestro modelo en el futuro conducirá a estimaciones demasiado optimistas. Por lo tanto, necesitamos recurrir a una división en datos de entrenamiento para la construcción del modelo, datos de validación para la selección del modelo y los parámetros, y datos de prueba para la evaluación del modelo. En lugar de una división simple, podemos reemplazar cada una de estas divisiones con validación cruzada. La forma más comúnmente utilizada (como se describió anteriormente) es una división de entrenamiento/prueba para la evaluación y el uso de la validación cruzada en el conjunto de entrenamiento para la selección del modelo y los parámetros.

El segundo punto se relaciona con la importancia de la métrica de evaluación o la función de puntuación utilizada para la selección y evaluación de modelos. La teoría sobre cómo tomar decisiones empresariales a partir de las predicciones de un modelo de aprendizaje automático queda fuera del alcance de este libro.7 Sin embargo, rara vez el objetivo final de una tarea de aprendizaje automático es construir un modelo con alta precisión. Asegúrese de que la métrica que elija para evaluar y seleccionar un modelo sea un buen sustituto de su uso real. En realidad, los problemas de clasificación rara vez tienen clases equilibradas, y los falsos positivos y los falsos negativos suelen tener consecuencias muy diferentes.

Recomendamos encarecidamente el libro de Foster Provost y Tom Fawcett Data Science for Business (O'Reilly) para obtener más información sobre este tema.

Asegúrese de comprender cuáles son estas consecuencias y elija una métrica de evaluación en consecuencia.

Las técnicas de evaluación y selección de modelos descritas hasta ahora son las herramientas más importantes del conjunto de herramientas de un científico de datos. La búsqueda en cuadrícula y la validación cruzada, tal como las hemos descrito en este capítulo, solo pueden aplicarse a un único modelo supervisado. Sin embargo, ya hemos visto que muchos modelos requieren preprocesamiento y que, en algunas aplicaciones, como el ejemplo de reconocimiento facial del capítulo 3, puede ser útil extraer una representación diferente de los datos. En el siguiente capítulo, presentaremos la clase Pipeline , que permite utilizar la búsqueda en cuadrícula y la validación cruzada en estas complejas cadenas de algoritmos.

Curvas de aprendizaje. Si realiza una regresión polinómica de alto grado, probablemente ajustará los datos de entrenamiento mucho mejor que con una regresión lineal simple. Por ejemplo, la Figura 4-14 aplica un modelo polinómico de 300 grados a los datos de entrenamiento anteriores y compara el resultado con un modelo lineal puro y un modelo cuadrático (polinomio de segundo grado). Observe cómo el modelo polinómico de 300 grados se mueve para acercarse lo más posible a las instancias de entrenamiento.

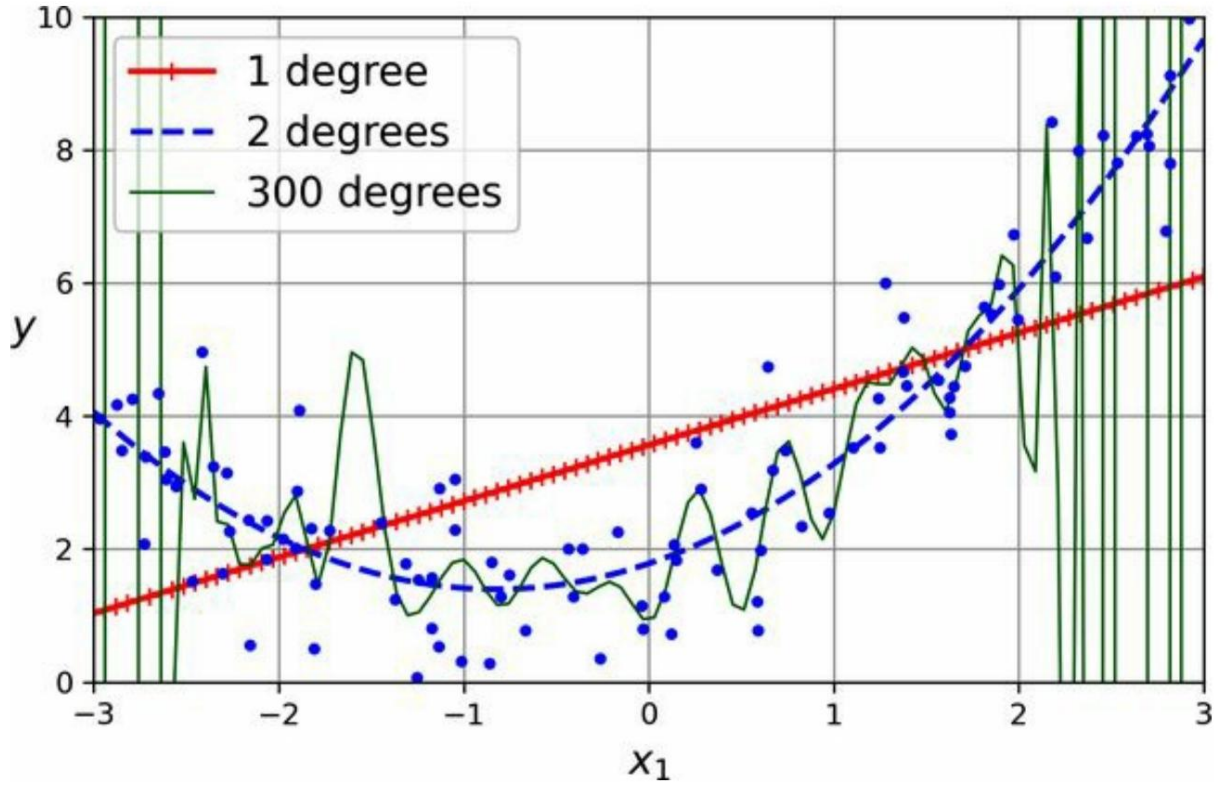

Figura 4-14. Regresión polinómica de alto grado

Este modelo de regresión polinómica de alto grado sobreajusta considerablemente los datos de entrenamiento, mientras que el modelo lineal los subajusta. El modelo que mejor se generaliza en este caso es el modelo cuadrático, lo cual tiene sentido porque los datos se generaron utilizando un modelo cuadrático. Sin embargo, en general, no se sabe qué función generó los datos, así que ¿cómo se puede determinar la complejidad del modelo? ¿Cómo se puede determinar si el modelo sobreajusta o subajusta los datos?

En el capítulo 2, se utilizó la validación cruzada para obtener una estimación del rendimiento de generalización de un modelo. Si un modelo tiene un buen rendimiento con los datos de entrenamiento, pero generaliza mal según las métricas de validación cruzada, se trata de un modelo sobreajustado. Si su rendimiento es bajo en ambos casos, se trata de un modelo subajustado. Esta es una forma de determinar si un modelo es demasiado simple o demasiado complejo.

Otra forma de determinarlo es observar las curvas de aprendizaje, que son gráficos del error de entrenamiento y el error de validación del modelo en función de la iteración de entrenamiento: simplemente evalúe el modelo a intervalos regulares durante el entrenamiento, tanto en el conjunto de entrenamiento como en el de validación, y grafique los resultados. Si el modelo no se puede entrenar incrementalmente (es decir, si no admite partial_fit() ni warm_start), deberá entrenarlo varias veces en subconjuntos gradualmente mayores del conjunto de entrenamiento.

Scikit-Learn cuenta con la útil función learning_curve() para facilitar este proceso: entrena y evalúa el modelo mediante validación cruzada. Por defecto, reentrena el modelo con subconjuntos crecientes del conjunto de entrenamiento, pero si el modelo admite aprendizaje incremental, se puede establecer exploit_incremental_learning=True al llamar a learning_curve() y, en su lugar, entrenará el modelo de forma incremental. La función devuelve los tamaños de los conjuntos de entrenamiento con los que se evaluó el modelo, así como las puntuaciones de entrenamiento y validación medidas para cada tamaño y para cada pliegue de validación cruzada. Utilicemos esta función para observar las curvas de aprendizaje del modelo de regresión lineal simple (véase la Figura 4-15).

```
de sklearn.model_selection importar curva_de_aprendizaje

tamaños_de_tren, puntuaciones_de_tren, puntuaciones_válidas =
    curva_de_aprendizaje( Regresión_lineal(), X, y, tamaños_de_tren=np.linspace(0.01, 1.0,
    40), cv=5, puntuación="error_cuadrático_medio_negativo")
errores_de_entrenamiento =
-puntuaciones_de_entrenamiento.media(eje=1) errores_válidos = -puntuaciones_válidas.media(eje=1)

plt.plot(train_sizes, train_errors, "r-+", linewidth=2, label="train") plt.plot(train_sizes,
valid_errors, "b-", linewidth=3, label="valid") [...] # embellecer la figura: agregar
etiquetas, eje, cuadrícula y leyenda plt.show()
```

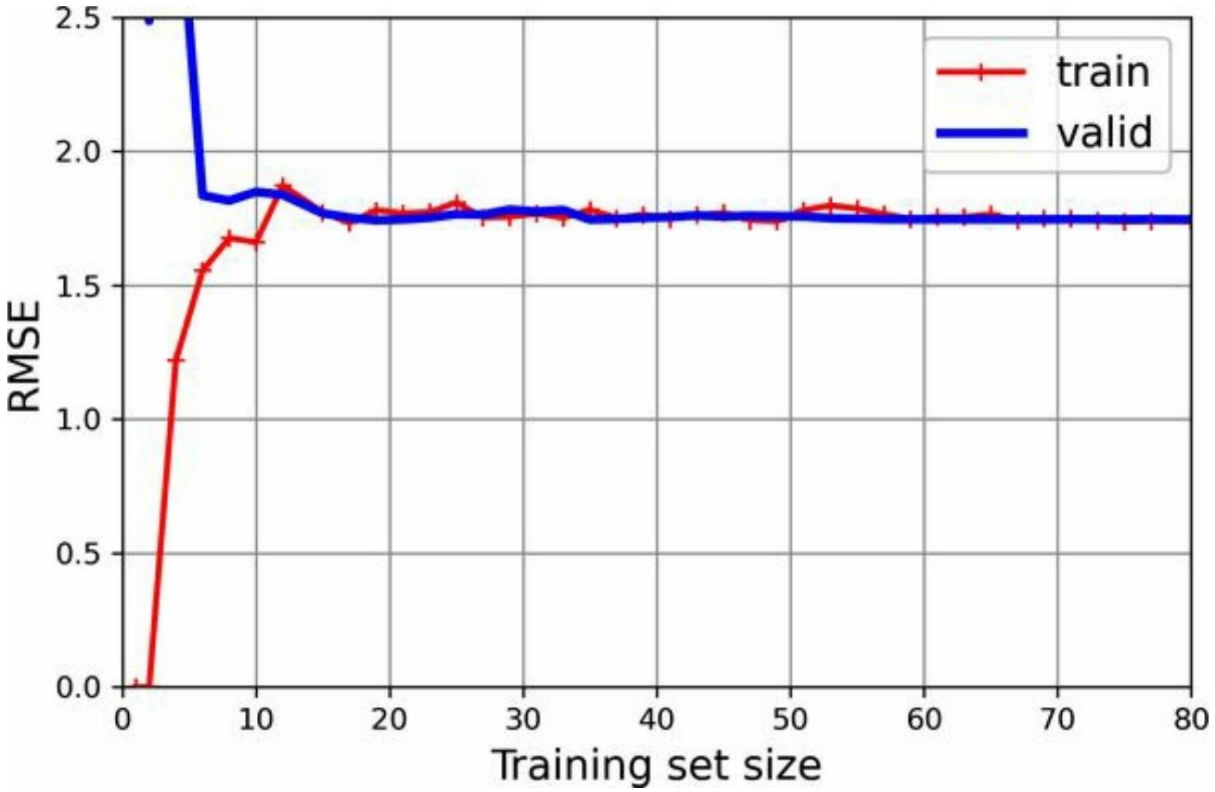

Figura 4-15. Curvas de aprendizaje

Este modelo presenta un subajuste. Para entender por qué, primero veamos el error de entrenamiento. Cuando solo hay una o dos instancias en el conjunto de entrenamiento, el modelo puede ajustarlas perfectamente, por lo que la curva comienza en cero. Sin embargo, a medida que se añaden nuevas instancias al conjunto de entrenamiento, al modelo le resulta imposible ajustarse perfectamente a los datos de entrenamiento, tanto por el ruido de los datos como por su falta de linealidad. Por lo tanto, el error en los datos de entrenamiento aumenta hasta alcanzar una meseta, momento en el que añadir nuevas instancias al conjunto de entrenamiento no mejora ni empeora significativamente el error promedio. Analicemos ahora el error de validación.

Cuando el modelo se entrena con muy pocas instancias de entrenamiento, no puede generalizar correctamente, por lo que el error de validación es inicialmente bastante grande. Luego, a medida que se le muestran más ejemplos de entrenamiento, el modelo aprende y, por lo tanto, el error de validación disminuye gradualmente. Sin embargo, una vez más, una línea recta no puede modelar bien los datos, por lo que el error se estanca, muy cerca de la otra curva.

Estas curvas de aprendizaje son típicas de un modelo con subajuste. Ambas curvas se han estancado; son cercanas y bastante altas.

> CONSEJO
>
> Si su modelo no se ajusta a los datos de entrenamiento, añadir más ejemplos de entrenamiento no servirá de nada. Necesita usar un modelo mejor o desarrollar mejores características.

Ahora veamos las curvas de aprendizaje de un modelo polinomial de décimo grado en los mismos datos (Figura 4-16):

```
desde sklearn.pipeline importar make_pipeline

regresión polinomial = crear_tubería(
    Características polinomiales (grado=10, incluir_sesgo=Falso),
    Regresión lineal())

tamaños_de_tren, puntuaciones_de_tren, puntuaciones_válidas = curva_de_aprendizaje(
    regresión polinomial, X, y, tamaños de tren=np.linspace(0.01, 1.0, 40), cv=5,
    puntuación="error cuadrático medio negativo")
[...] # lo mismo que antes
```

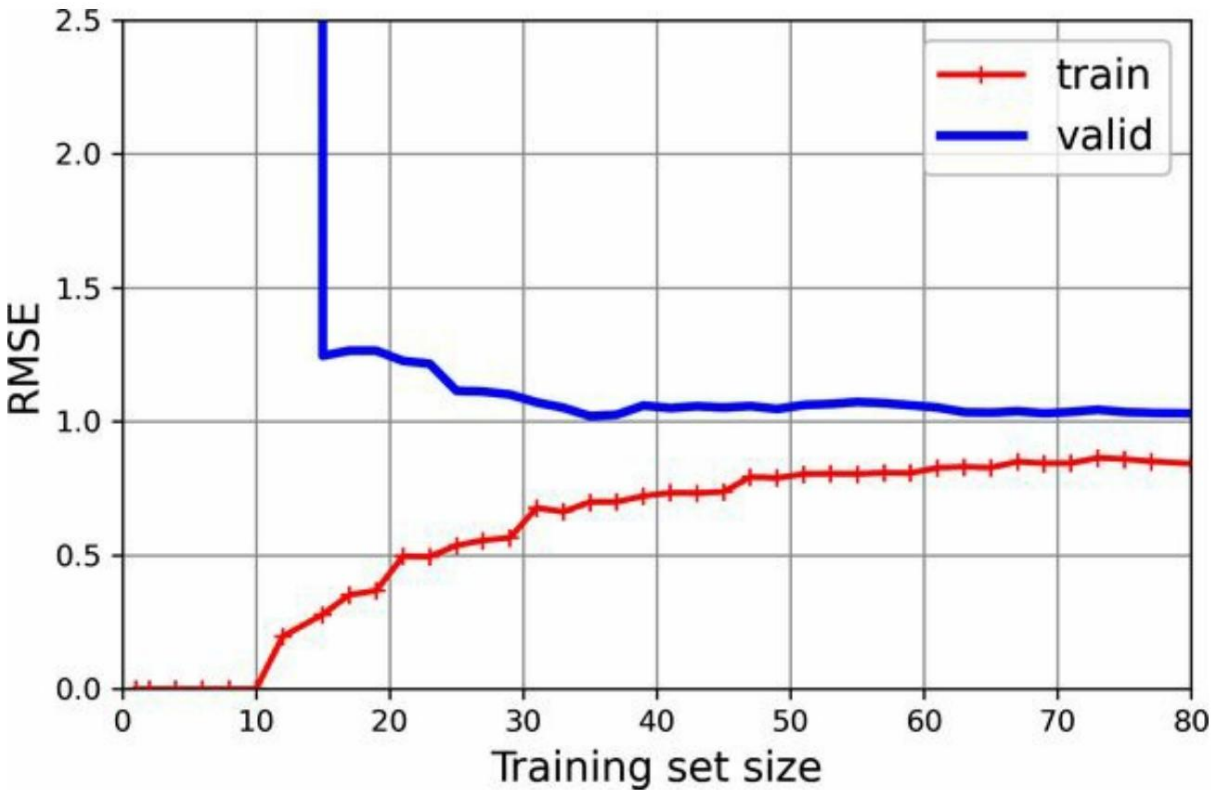

Figura 4-16. Curvas de aprendizaje para el modelo polinomial de décimo grado

Estas curvas de aprendizaje se parecen un poco a las anteriores, pero hay dos diferencias muy importantes:

- El error en los datos de entrenamiento es mucho menor que antes.
- Existe una brecha entre las curvas. Esto significa que el modelo funciona significativamente mejor con los datos de entrenamiento que con los de validación, lo cual es característico de un modelo sobreajustado. Sin embargo, si se utilizara un conjunto de entrenamiento mucho mayor, las dos curvas seguirían acercándose.

CONSEJO

Una forma de mejorar un modelo sobreajustado es alimentarlo con más datos de entrenamiento hasta que el error de validación alcance el error de entrenamiento.

EL COMPROMISO ENTRE SESGO Y VARIANZA

Un resultado teórico importante de la estadística y el aprendizaje automático es el hecho de que el error de generalización de un modelo puede expresarse como la suma de tres errores muy diferentes:

Inclinación

Esta parte del error de generalización se debe a suposiciones erróneas, como suponer que los datos son lineales cuando en realidad son cuadráticos.
Es muy probable que un modelo con alto sesgo no se ajuste a los datos de entrenamiento. 6

Diferencia

Esta parte se debe a la excesiva sensibilidad del modelo a pequeñas variaciones en los datos de entrenamiento. Un modelo con muchos grados de libertad (como un modelo polinomial de alto grado) probablemente presente una alta varianza y, por lo tanto, sobreajuste los datos de entrenamiento.

Error irreducible

Esta parte se debe al ruido de los propios datos. La única forma de...

La forma de reducir esta parte del error es limpiar los datos (por ejemplo, arreglar las fuentes de datos, como sensores rotos, o detectar y eliminar valores atípicos).

Aumentar la complejidad de un modelo generalmente aumenta su varianza y reduce su sesgo. Por el contrario, reducir la complejidad de un modelo aumenta su sesgo y reduce su varianza. Por eso se denomina compensación.

Suponiendo un error distribuido gaussiano y maximizando la verosimilitud, se minimiza la suma de los errores al cuadrado. Otra medida es la error cuadrático relativo (RSE):

$$\text{ERSE} = \frac{\sum_t [r^t - g(x^t|\theta)]^2}{\sum_t (r^t - \bar{r})^2} \tag{4.34}$$

Si ERSE se acerca a 1, nuestra predicción es tan buena como la del promedio; a medida que se acerca a 0, mejor ajuste. Si ERSE se acerca a 1, significa que usar un modelo basado en la entrada x no funciona mejor que usar el promedio, que sería nuestro estimador si no existiera x; si ERSE se acerca a 0, la entrada x es útil.

coeficiente de determinación

Para comprobar si la regresión se ajusta bien, una medida es el coeficiente de determinación que es

$$R^2 = 1 - \text{ERSE}$$

y para que la regresión se considere útil, requerimos que R^2 esté cerca de 1.

Recuerde que para una generalización óptima, debemos ajustar la complejidad de nuestro modelo de aprendizaje a la complejidad de los datos. En la regresión polinómica, el parámetro de complejidad es el orden del polinomio ajustado; por lo tanto, debemos encontrar la manera de elegir el orden óptimo que minimice el error de generalización; es decir, ajustar la complejidad del modelo para que se ajuste mejor a la complejidad de la función inherente a los datos.

4.7 Ajuste de la complejidad del modelo: dilema sesgo/varianza

Supongamos que se extrae una muestra $\mathcal{X}=\{x^t, r^t\}$ de una densidad de probabilidad conjunta desconocida $p(x, r)$. Con esta muestra, construimos nuestra estimación $g(\ )$. El error cuadrático esperado (sobre la densidad conjunta) en x puede escribirse como (usando la ecuación 4.17)

$$E[(r - g(x))^2|x] = \underbrace{E[(r - E[r|x])^2|x]}_{\text{ruido}} + \underbrace{(E[r|x] - g(x))^2}_{\text{cuadrado ed er r o}} \tag{4.35}$$

El primer término a la derecha es la varianza de r dado x; no depende de $g(\ )$ ni de X. Es la varianza del ruido añadido, σ^2. Esta es la parte del error que nunca se puede eliminar, independientemente del estimador que utilicemos.
El segundo término cuantifica cuánto se desvía g(x) de la función de regresión, E[r|x]. Esto depende del estimador y del conjunto de entrenamiento.

Puede darse el caso de que, para una muestra, g(x) tenga un ajuste muy bueno; y para otra, un ajuste deficiente. Para cuantificar la eficacia de un estimador g( ) , se promedian los posibles conjuntos de datos.

El valor esperado (promedio de las muestras X, todas de tamaño N y extraídas) de la misma densidad conjunta p(r, x)) es (usando la ecuación 4.11)

$$EX[(E[r|x]-g(x))^2|x] = (E[r|x] - EX[g(x)])^2 + EX[(g(x) - EX[g(x)])^2] \tag{4.36}$$

inclinación varianza

Como comentamos antes, el sesgo mide cuánto g(x) es incorrecto sin tener en cuenta el efecto de las muestras variables, y la varianza mide cuánto fluctúa g(x) alrededor del valor esperado, E[g(x)], a medida que la muestra varía.
Queremos que ambos sean pequeños.

Veamos un ejemplo didáctico: Para estimar el sesgo y la varianza, rt }, i = 1,...,M, a partir de Generamos varios conjuntos de datos Xi = {xt conocido f_i , unos ( ) con ruido añadido, utilizamos cada conjunto de datos para formar un estimador gi( ) y calculamos el sesgo y la varianza. Cabe destacar que, en la práctica, esto no es posible porque desconocemos f ( ) ni los parámetros del ruido añadido. Entonces, E[g(x)] se estima mediante el promedio sobre gi( ):

$$\bar{g}(x) = \frac{1}{M}\sum_{i=1}^{METRO} gi(x)$$

El sesgo y la varianza estimados son

$$\text{Sesgo}^2(g) = \frac{1}{norte}\sum_{el}[\bar{g}(xt) - f(xt)]^2$$

$$\text{Varianza}(g) = \frac{1}{Nuevo\ Méjico}\sum_{el\ i}[gi(xt) - \bar{g}(xt)]^2$$

Veamos algunos modelos de diferente complejidad: El más simple es un ajuste constante.

$$gi(x) = 2$$

Esto no tiene varianza porque no utilizamos los datos y todos los gi(x) son iguales. Sin embargo, el sesgo es alto, a menos que, por supuesto, f(x) sea cercano a 2 para todos los x. Si tomamos el promedio de rt en la muestra

$$gi(x) = \sum_{el} rt_i /NORTE$$

En lugar de la constante 2, esto disminuye el sesgo porque esperaríamos que el promedio en general fuera una mejor estimación. Pero esto aumenta el

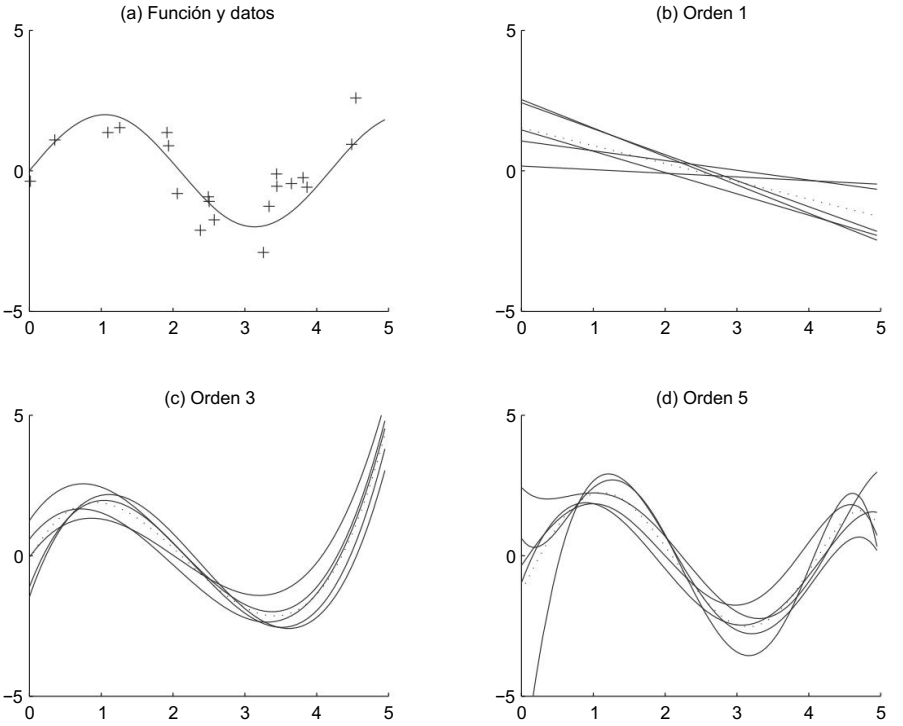

Figura 4.5 (a) Función, f (x) = 2 sin(1.5x), y un conjunto de datos ruidoso (N(0, 1))
Muestreo de la función. Se toman cinco muestras, cada una con veinte instancias. (b), (c) y (d) son cinco ajustes polinómicos, a saber, gi( ), de orden 1, 3 y 5.
Para cada caso, la línea de puntos es el promedio de los cinco ajustes, es decir, g(¯).

varianza porque las diferentes muestras Xi tendrían promedios diferentes valores. Normalmente, en este caso, la disminución del sesgo sería mayor que El aumento de la varianza y el error disminuirían.

En el contexto de la regresión polinómica, se da un ejemplo en la figura 4.5. A medida que aumenta el orden del polinomio, pequeños cambios en la El conjunto de datos provoca un cambio mayor en los polinomios ajustados; por lo tanto, la varianza aumenta. Pero un modelo complejo en promedio permite un mejor ajuste a la función subyacente; por lo tanto, el sesgo disminuye (véase la figura 4.6). Esto se denomina Sesgo/varianza El dilema del sesgo/varianza es cierto para cualquier sistema de aprendizaje automático. dilema y no solo para la regresión polinómica (Geman, Bienenstock y Dour-sat 1992). Para reducir el sesgo, el modelo debe ser flexible, a riesgo de

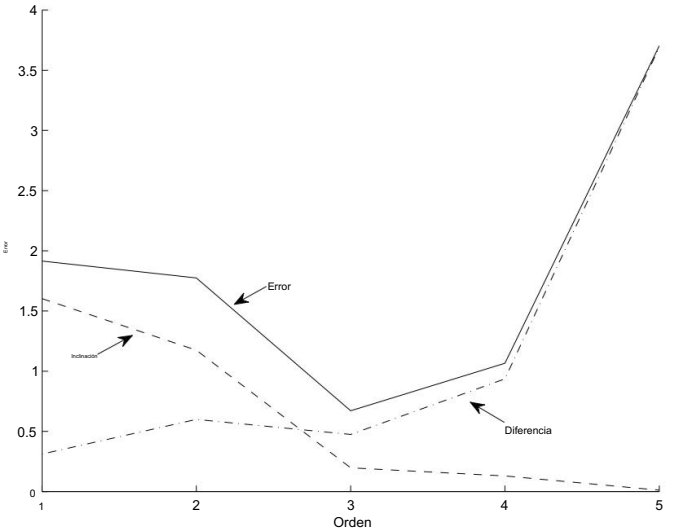

Figura 4.6 En el mismo contexto que la figura 4.5, utilizando cien modelos en lugar de cinco, sesgo, varianza y error para polinomios de orden 1 a 5. El orden 1 presenta la varianza más baja. El orden 5 presenta el sesgo más bajo. A medida que aumenta el orden, el sesgo disminuye, pero la varianza aumenta. El orden 3 presenta el error mínimo.

Presentan una alta varianza. Si la varianza se mantiene baja, es posible que no logremos un buen ajuste a los datos y que tengamos un sesgo elevado. El modelo óptimo es aquel que ofrece el mejor equilibrio entre el sesgo y la varianza.

Si hay sesgo, esto indica que nuestra clase de modelo no contiene subajuste de la solución; esto es subajuste. Si hay varianza, la clase de modelo está sobreajustando demasiado general y también aprende el ruido; esto es sobreajuste. Si g( ) es de la misma clase de hipótesis que f ( ), por ejemplo, un polinomio del mismo orden, tenemos un estimador insesgado y el sesgo estimado disminuye a medida que aumenta el número de modelos. Esto muestra el efecto de reducción de error de elegir el modelo correcto (que llamamos sesgo inductivo en el capítulo 2; los dos usos de "sesgo" son diferentes pero no inconexos). En cuanto a la varianza, también depende del tamaño del conjunto de entrenamiento; la variabilidad debida a la muestra disminuye a medida que aumenta el tamaño de la muestra. En resumen, para obtener un valor pequeño de error, debemos tener el sesgo inductivo adecuado (para obtener un sesgo pequeño en el sentido estadístico) y tener un conjunto de datos lo suficientemente grande como para que la variabilidad del modelo pueda restringirse con los datos.

Tenga en cuenta que cuando la varianza es grande, el sesgo es bajo: esto indica que $g(\bar{x})$ es un buen estimador. Por lo tanto, para obtener un valor de error pequeño, podemos tomar un valor grande. número de modelos de alta varianza y utilizamos su promedio como nuestro estimador. Discutiremos estos enfoques para la combinación de modelos en el capítulo 17.

4.8 Procedimientos de selección de modelos

Hay una serie de procedimientos que podemos utilizar para ajustar la complejidad del modelo.

validación cruzada

En la práctica, el método que utilizamos para encontrar la complejidad óptima es la validación cruzada. No podemos calcular el sesgo ni la varianza de un modelo, pero sí podemos... Calcular el error total. Dado un conjunto de datos, lo dividimos en dos partes como conjuntos de entrenamiento y validación, entrenan modelos candidatos de diferentes complejidades y prueban su error en el conjunto de validación omitido durante el entrenamiento. A medida que aumenta la complejidad del modelo, el error de entrenamiento sigue disminuyendo. El error en el conjunto de validación disminuye hasta un cierto nivel de complejidad, luego deja de disminuir o no disminuye más significativamente, o incluso aumenta si hay ruido significativo. Este "codo" corresponde a la nivel de complejidad óptimo (ver figura 4.7).

En la vida real, no podemos calcular el sesgo y, por lo tanto, el error como lo hacemos en la figura 4.6; el error de validación en la figura 4.7 es una estimación de eso, excepto que también contiene ruido: Incluso si tenemos el modelo correcto que hay Si no hay sesgo y los datos son lo suficientemente grandes como para que la varianza sea insignificante, puede haber Aún así, el error de validación no es cero. Nótese que el error de validación de la figura 4.7 no tiene la misma forma de V que el error de la figura 4.6 porque el primero utiliza más datos de entrenamiento y sabemos que podemos restringir la varianza con más datos. De hecho, vemos en la figura 4.5(d) que incluso el polinomio de quinto orden se comporta como uno de tercer orden cuando hay datos; por ejemplo, en el dos extremos donde hay menos puntos de datos, no es tan preciso.

regularización

Otro enfoque que se utiliza con frecuencia es la regularización (Breiman 1998). En este enfoque, escribimos una función de error aumentada

$$E \text{ (4.37)} = \text{error en los datos} + \lambda \quad \text{complejidad del modelo}$$

Esto tiene un segundo término que penaliza los modelos complejos con gran varianza, donde λ indica el peso de esta penalización. Cuando minimizamos el Al usar la función de error aumentada en lugar del error basado únicamente en los datos, penalizamos los modelos complejos y, por lo tanto, reducimos la varianza. Si λ es demasiado grande, Sólo se permiten modelos muy simples y corremos el riesgo de introducir sesgos. λ es Optimizado mediante validación cruzada.