Esto se parece bien a California, pero aparte de eso, es difícil ver un patrón en particular. Establecer el `alfa` opción a `0,1` hace que sea mucho más fácil visualizar los lugares donde hay una alta densidad de puntos de datos ( Figura 2-12 ):

```
alojamiento . trama ( tipo = "dispersión" , X = "longitud" , y = "latitud" , alfa = 0,1 )
```

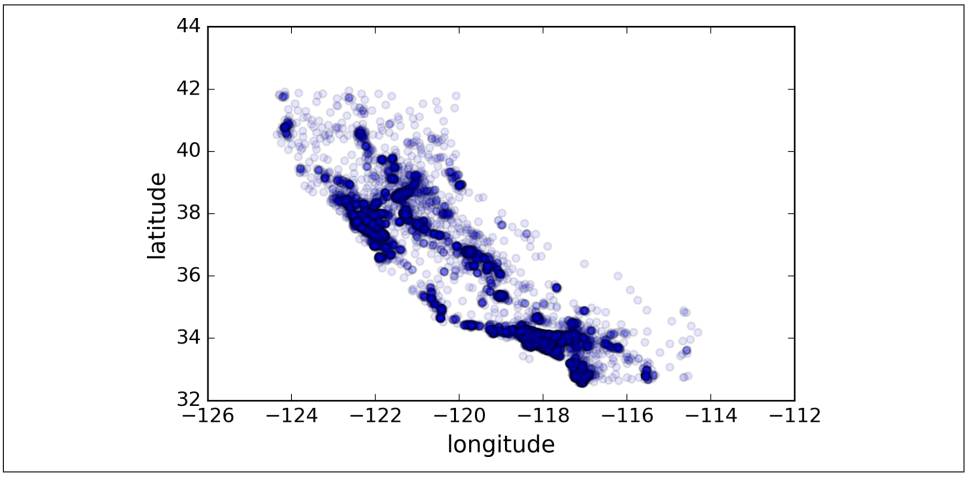


*Figura 2-12. Una mejor visualización destacando áreas de alta densidad.*

Ahora eso es mucho mejor: puede ver claramente las áreas de alta densidad, a saber, el Área de la Bahía y alrededor de Los Ángeles y San Diego, además de una larga fila de densidad bastante alta en el Valle Central, en particular alrededor de Sacramento y Fresno.

De manera más general, nuestros cerebros son muy buenos para detectar patrones en imágenes, pero es posible que deba jugar con los parámetros de visualización para que los patrones se destaquen.

Ahora veamos los precios de la vivienda ( Figura 2-13 ). El radio de cada círculo representa la población del distrito (opción `s`), y el color representa el precio (opción `C`). Usaremos un mapa de colores predefinido (opción `cmap`) llamado `chorro`, que va del azul (valores bajos) al rojo (precios altos): [15]

```
alojamiento . trama ( tipo = "dispersión" , X = "longitud" , y = "latitud" , alfa = 0.4 ,
        s = alojamiento [ "población" ] / 100 , etiqueta = "población" ,
        C = "valor_median_house" , cmap = plt . get_cmap ( "chorro" ), barra de color = Cierto ,
)
plt . leyenda ()
```

15 Si está leyendo esto en escala de grises, tome un bolígrafo rojo y garabatee sobre la mayor parte de la costa del Área de la Bahía hasta San Diego (como era de esperar). También puede agregar un parche de amarillo alrededor de Sacramento.

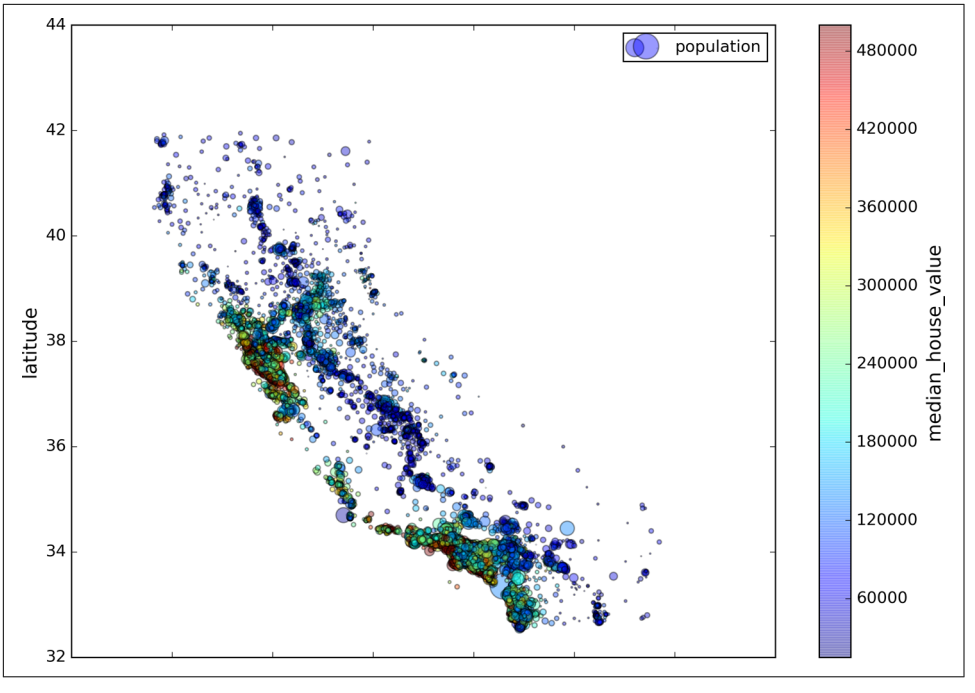


*Figura 2-13. Precios de la vivienda en California*

Esta imagen le dice que los precios de la vivienda están muy relacionados con la ubicación (por ejemplo, cerca del océano) y con la densidad de población, como probablemente ya lo sabía. Probablemente será útil utilizar un algoritmo de agrupación para detectar las agrupaciones principales y agregar nuevas características que midan la proximidad a los centros de la agrupación. El atributo de proximidad del océano también puede ser útil, aunque en el norte de California los precios de la vivienda en los distritos costeros no son demasiado altos, por lo que no es una regla simple.

## Buscando correlaciones

Dado que el conjunto de datos no es demasiado grande, puede calcular fácilmente el *coeficiente de correlación estándar (* también llamado *R de Pearson* entre cada par de atributos usando el corr ()
método:

```
corr_matrix = alojamiento . corr ()
```

Ahora veamos cuánto se correlaciona cada atributo con el valor medio de la vivienda:

```
>>>  corr_matrix [ "valor_median_house" ] . sort_values ( ascendente = Falso )
median_house_value          1.000000
ingreso medio               0,687170
total_rooms                 0.135231
vivienda_median_age         0.114220
hogares                     0.064702
```

| | |
|---|---|
| total_drooms | 0.047865 |
| población | - 0.026699 |
| longitud | - 0.047279 |
| latitud | - 0.142826 |

Nombre: median_house_value, dtype: float64

El coeficiente de correlación varía de –1 a 1. Cuando se acerca a 1, significa que hay una fuerte correlación positiva; por ejemplo, el valor medio de la vivienda tiende a aumentar cuando aumenta el ingreso medio. Cuando el coeficiente se acerca a –1, significa que existe una fuerte correlación negativa; puede ver una pequeña correlación negativa entre la latitud y el valor medio de la vivienda (es decir, los precios tienen una ligera tendencia a bajar cuando va al norte). Finalmente, coeficientes cercanos a cero significan que no existe correlación lineal. Figura 2-14 muestra varias gráficas junto con el coeficiente de correlación entre sus ejes horizontal y vertical.

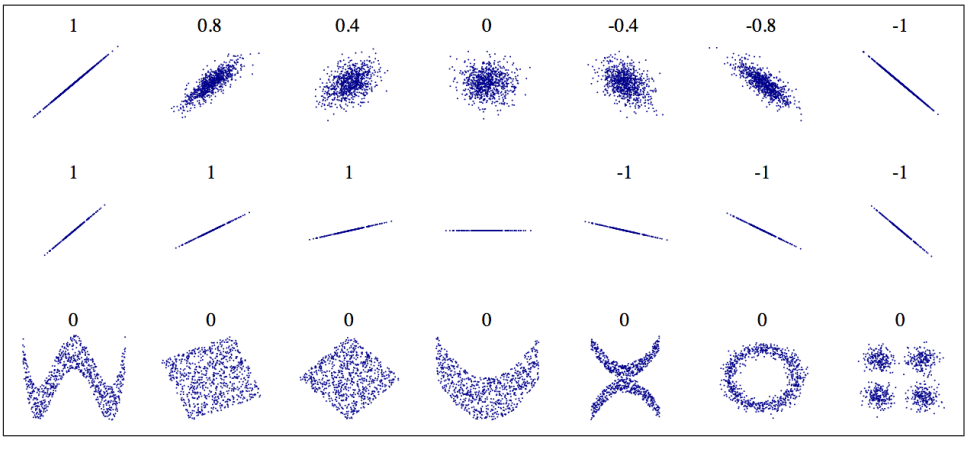


*Figura 2-14. Coeficiente de correlación estándar de varios conjuntos de datos (fuente: Wikipedia; imagen de dominio público)*

El coeficiente de correlación solo mide correlaciones lineales ("si *X* sube, entonces *y* generalmente sube / baja "). Puede perder completamente las relaciones no lineales (por ejemplo, "si *X* está cerca de cero entonces *y* generalmente sube "). Observe cómo todas las gráficas de la fila inferior tienen un coeficiente de correlación igual a cero a pesar de que sus ejes claramente no son independientes: estos son ejemplos de relaciones no lineales. Además, la segunda fila muestra ejemplos en los que el coeficiente de correlación es igual a 1 o –1; observe que esto no tiene nada que ver con la pendiente. Por ejemplo, su altura en pulgadas tiene un coeficiente de correlación de 1 con su altura en pies o en nanómetros.

Otra forma de verificar la correlación entre atributos es usar Pandas ' scatter_matrix función, que traza cada atributo numérico contra todos los demás atributos numéricos. Como ahora hay 11 atributos numéricos, obtendría $11^2 =$

121 parcelas, que no cabrían en una página, así que centrémonos en algunos atributos prometedores que parecen estar más correlacionados con el valor medio de la vivienda ( Figura 2-15 ):

```
de pandas.tools.plotting importar scatter_matrix

atributos = [ "valor_median_house" , "ingreso medio" , "total_rooms" ,
                    "vivienda_median_age" ]
scatter_matrix ( alojamiento [ atributos ], figsize = ( 12 , 8 ))
```

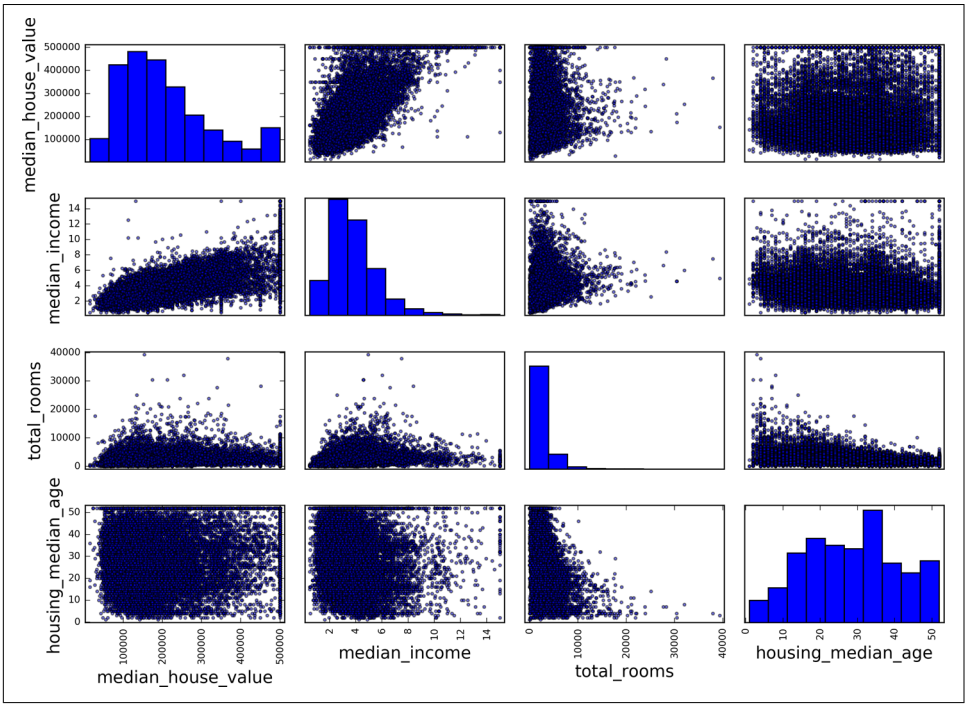


*Figura 2-15. Matriz de dispersión*

La diagonal principal (de arriba a la izquierda a abajo a la derecha) estaría llena de líneas rectas si Pandas trazara cada variable contra sí misma, lo que no sería muy útil. Entonces, en su lugar, Pandas muestra un histograma de cada atributo (hay otras opciones disponibles; consulte la documentación de Pandas para obtener más detalles).

El atributo más prometedor para predecir el valor medio de la vivienda es el ingreso medio, así que vamos a ampliar su diagrama de dispersión de correlación ( Figura 2-16 ):

```
alojamiento . trama ( tipo = "dispersión" , X = "ingreso medio" , y = "valor_median_house" ,
                    alfa = 0,1 )
```

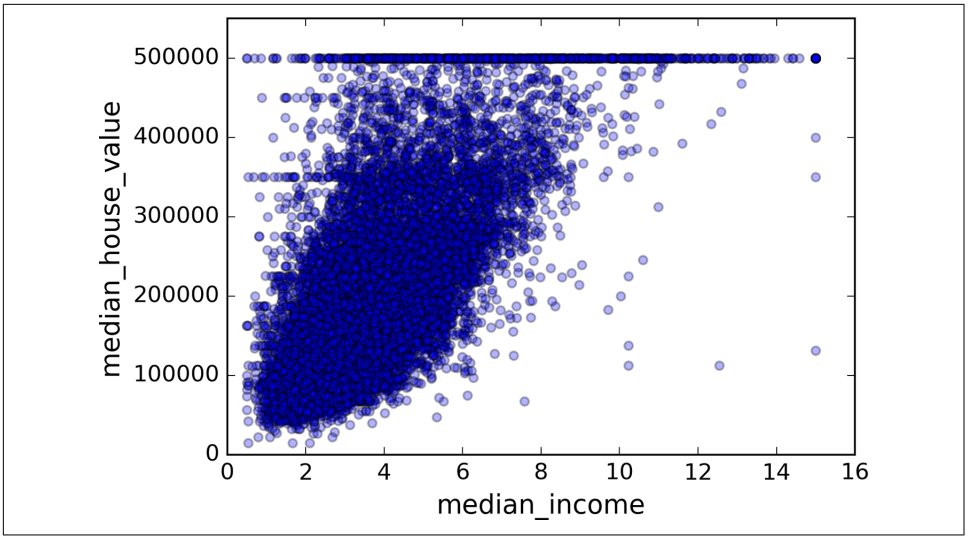


*Figura 2-16. Ingresos medios frente al valor medio de la vivienda*

Esta trama revela algunas cosas. Primero, la correlación es realmente muy fuerte; se puede ver claramente la tendencia alcista y los puntos no están demasiado dispersos. En segundo lugar, el límite de precio que notamos anteriormente es claramente visible como una línea horizontal en $ 500,000. Pero esta gráfica revela otras líneas rectas menos obvias: una línea horizontal alrededor de $ 450,000, otra alrededor de $ 350,000, quizás una alrededor de $ 280,000 y algunas más por debajo de eso. Quizás desee intentar eliminar los distritos correspondientes para evitar que sus algoritmos aprendan a reproducir estas peculiaridades de los datos.

## Experimentar con combinaciones de atributos

Con suerte, las secciones anteriores le dieron una idea de algunas formas en que puede explorar los datos y obtener información. Identificó algunas peculiaridades de los datos que quizás desee limpiar antes de enviar los datos a un algoritmo de aprendizaje automático, y encontró correlaciones interesantes entre los atributos, en particular con el atributo de destino. También notó que algunos atributos tienen una distribución de cola pesada, por lo que es posible que desee transformarlos (por ejemplo, calculando su logaritmo). Por supuesto, su kilometraje variará considerablemente con cada proyecto, pero las ideas generales son similares.

Una última cosa que puede querer hacer antes de preparar los datos para los algoritmos de Machine Learning es probar varias combinaciones de atributos. Por ejemplo, el número total de habitaciones en un distrito no es muy útil si no sabe cuántos hogares hay. Lo que realmente desea es la cantidad de habitaciones por hogar. Del mismo modo, el número total de dormitorios por sí solo no es muy útil: probablemente desee compararlo con el número de habitaciones. Y la población por hogar también

parece una combinación de atributos interesante a la vista. Creemos estos nuevos atributos:

```
alojamiento [ "habitaciones_por_familia" ] = alojamiento [ "total_rooms" ] / alojamiento [ "hogares" ]
alojamiento [ "dormitorios_por_habitación" ] = alojamiento [ "total_drooms" ] / alojamiento [ "total_rooms" ]
alojamiento [ "población_por_familia" ] = alojamiento [ "población" ] / alojamiento [ "hogares" ]
```

Y ahora veamos la matriz de correlación nuevamente:

```
>>> corr_matrix = alojamiento . corr ()
>>> corr_matrix [ "valor_median_house" ] . sort_values ( ascendente = Falso )
median_house_value 1.000000
ingreso medio 0,687170
rooms_per_household 0.199343
total_rooms 0.135231
vivienda_median_age 0.114220
hogares 0.064702
total_drooms 0.047865
población_por_familia - 0.021984
población - 0.026699
longitud - 0.047279
latitud - 0.142826
dormitorios_por_habitación - 0,260070
Nombre: median_house_value, dtype: float64
```

¡Oye, no está mal! El nuevo dormitorios_por_habitación El atributo está mucho más correlacionado con el valor medio de la vivienda que con el número total de habitaciones o dormitorios. Aparentemente, las casas con una relación dormitorio / habitación más baja tienden a ser más caras. El número de habitaciones por hogar también es más informativo que el número total de habitaciones en un distrito; obviamente, cuanto más grandes son las casas, más caras son.

Esta ronda de exploración no tiene por qué ser absolutamente exhaustiva; el punto es comenzar con el pie derecho y obtener rápidamente conocimientos que lo ayudarán a obtener un primer prototipo razonablemente bueno. Pero este es un proceso iterativo: una vez que tiene un prototipo en funcionamiento, puede analizar su salida para obtener más información y volver a este paso de exploración.

## Prepare los datos para algoritmos de aprendizaje automático

Es hora de preparar los datos para sus algoritmos de aprendizaje automático. En lugar de simplemente hacer esto manualmente, debe escribir funciones para hacer eso, por varias buenas razones:

- Esto le permitirá reproducir estas transformaciones fácilmente en cualquier conjunto de datos (por ejemplo, la próxima vez que obtenga un conjunto de datos nuevo).
- Construirá gradualmente una biblioteca de funciones de transformación que podrá reutilizar en proyectos futuros.
- Puede utilizar estas funciones en su sistema en vivo para transformar los nuevos datos antes de alimentarlos a sus algoritmos.

- Esto le permitirá probar fácilmente varias transformaciones y ver qué combinación de transformaciones funciona mejor.

Pero primero volvamos a un conjunto de entrenamiento limpio (copiando strat_train_set una vez más), y separemos los predictores y las etiquetas, ya que no necesariamente queremos aplicar las mismas transformaciones a los predictores y los valores objetivo (tenga en cuenta que soltar()
crea una copia de los datos y no afecta strat_train_set):

```
alojamiento = strat_train_set . soltar ( "valor_median_house" , eje = 1 )
viviendas_etiquetas = strat_train_set [ "valor_median_house" ] . Copiar ()
```

## Limpieza de datos

La mayoría de los algoritmos de aprendizaje automático no pueden funcionar con las características que faltan, así que creemos algunas funciones para cuidarlas. Notaste antes que el total_drooms
El atributo tiene algunos valores faltantes, así que arreglemos esto. Tienes tres opciones:

- Deshazte de los distritos correspondientes.
- Deshazte de todo el atributo.
- Establezca los valores en algún valor (cero, la media, la mediana, etc.).

Puede lograr esto fácilmente usando DataFrame's dropna (), drop (), y fillna ()
métodos:

```
alojamiento . dropna ( subconjunto = [ "total_drooms" ])        # Opción 1
alojamiento . soltar ( "total_drooms" , eje = 1 )               # opcion 2
mediana = alojamiento [ "total_drooms" ] . mediana ()
alojamiento [ "total_drooms" ] . Fillna ( mediana )             # opción 3
```

Si elige la opción 3, debe calcular el valor de la mediana en el conjunto de entrenamiento y usarlo para completar los valores que faltan en el conjunto de entrenamiento, pero tampoco olvide guardar el valor de la mediana que ha calculado. Lo necesitará más adelante para reemplazar los valores faltantes en el conjunto de prueba cuando desee evaluar su sistema, y también una vez que el sistema entre en funcionamiento para reemplazar los valores faltantes en nuevos datos.

Scikit-Learn proporciona una clase útil para ocuparse de los valores faltantes: Imputer. He aquí cómo utilizarlo. Primero, necesitas crear un Imputer instancia, especificando que desea reemplazar los valores perdidos de cada atributo con la mediana de ese atributo:

```
de sklearn.preprocessing importar Imputer

imputador = Imputer ( estrategia = "mediana" )
```

Dado que la mediana solo se puede calcular en atributos numéricos, necesitamos crear una copia de los datos sin el atributo de texto proximidad_oceano:

```
número_vivienda = alojamiento . soltar ( "ocean_proximity" , eje = 1 )
```

Ahora puedes encajar el imputador instancia a los datos de entrenamiento usando el ajuste() método:

```
imputador . ajuste ( número_vivienda )
```

los imputador simplemente ha calculado la mediana de cada atributo y ha almacenado el resultado en su Estadísticas_ Instancia variable. Solo el total_drooms el atributo tenía valores perdidos, pero no podemos estar seguros de que no habrá ningún valor perdido en los datos nuevos después de que el sistema entre en funcionamiento, por lo que es más seguro aplicar el imputador a todos los atributos numéricos:

```
>>> imputador . Estadísticas_
matriz ([-118.51, 34.26, 29., 2119., 433., 1164., 408., 3.5414])
>>> número_vivienda . mediana () . valores
matriz ([-118.51, 34.26, 29., 2119., 433., 1164., 408., 3.5414])
```

Ahora puedes usar este "capacitado" imputador para transformar el conjunto de entrenamiento reemplazando los valores perdidos por las medianas aprendidas:

```
X = imputador . transformar ( número_vivienda )
```

El resultado es una matriz Numpy simple que contiene las características transformadas. Si desea volver a colocarlo en un Pandas DataFrame, es simple:

```
housing_tr = pd . Marco de datos ( X , columnas = número_vivienda . columnas )
```

## Diseño Scikit-Learn

La API de Scikit-Learn está muy bien diseñada. los principios principales de diseño son: dieciséis

- **Consistencia.** Todos los objetos comparten una interfaz simple y consistente:
  - *Estimadores*. Cualquier objeto que pueda estimar algunos parámetros en función de un conjunto de datos se denomina *estimador* por ejemplo, un imputador es un estimador). La estimación en sí la realiza el ajuste() método, y toma solo un conjunto de datos como parámetro (o dos para algoritmos de aprendizaje supervisado; el segundo conjunto de datos contiene las etiquetas). Cualquier otro parámetro necesario para guiar el proceso de estimación se considera un hiperparámetro (como un imputador es estrategia), y debe establecerse como una variable de instancia (generalmente a través de un parámetro de constructor).

  - *Transformadores*. Algunos estimadores (como un imputador) también puede transformar un conjunto de datos; estos se llaman *transformadores*. Una vez más, la API es bastante simple: la transformación la realiza el transformar() método con el conjunto de datos para transformar como parámetro. Devuelve el conjunto de datos transformado. Esta transformación generalmente se basa en los parámetros aprendidos, como es el caso de una imputador.

    Todos los transformadores también tienen un método de conveniencia llamado fit_transform ()

16 Para obtener más detalles sobre los principios de diseño, consulte “Diseño de API para software de aprendizaje automático: experiencias de el proyecto scikit-learn ”, L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Müller, et al. (2013).

eso es equivalente a llamar ajuste() y entonces transformar () ( pero a veces fit_transform () está optimizado y se ejecuta mucho más rápido).

- *Predictores.* Finalmente, algunos estimadores son capaces de hacer predicciones dado un conjunto de datos; se les llama *predictores.* Por ejemplo, el Regresión lineal El modelo del capítulo anterior fue un predictor: predijo la satisfacción con la vida dado el PIB per cápita de un país. Un predictor tiene un predecir() método que toma un conjunto de datos de nuevas instancias y devuelve un conjunto de datos de las predicciones correspondientes. También tiene un Puntuación() método que mide la calidad de las predicciones dado un conjunto de prueba (y las etiquetas correspondientes en el caso de algoritmos de aprendizaje supervisado). [17]

- **Inspección.** Todos los hiperparámetros del estimador son accesibles directamente a través de variables de instancia pública (por ejemplo, imputer.strategy), y todos los parámetros aprendidos del estimador también son accesibles a través de variables de instancia pública con un guión bajo sufijo (por ejemplo, imputer.statistics_).
- **No proliferación de clases.** Los conjuntos de datos se representan como matrices NumPy o matrices dispersas SciPy, en lugar de clases caseras. Los hiperparámetros son solo cadenas o números normales de Python.
- **Composición.** Los bloques de construcción existentes se reutilizan tanto como sea posible. Por ejemplo, es fácil crear un Tubería estimador de una secuencia arbitraria de transformadores seguida de un estimador final, como veremos.
- **Incumplimientos sensibles.** Scikit-Learn proporciona valores predeterminados razonables para la mayoría de los parámetros, lo que facilita la creación rápida de un sistema de trabajo básico.

## Manejo de texto y atributos categóricos

Anteriormente dejamos fuera el atributo categórico oceano_proximidad porque es un atributo de texto, por lo que no podemos calcular su mediana. La mayoría de los algoritmos de aprendizaje automático prefieren trabajar con números de todos modos, así que convierta estas etiquetas de texto en números.

Scikit-Learn proporciona un transformador para esta tarea llamado LabelEncoder:

```
>>> de sklearn.preprocessing importar LabelEncoder
>>> codificador = LabelEncoder ()
>>> vivienda_gato = alojamiento [ "ocean_proximity" ]
>>> Housing_cat_encoded = codificador . fit_transform ( vivienda_gato )
>>> Housing_cat_encoded
matriz ([1, 1, 4, ..., 1, 0, 3])
```

17 Algunos predictores también proporcionan métodos para medir la confianza de sus predicciones.

Esto es mejor: ahora podemos usar estos datos numéricos en cualquier algoritmo ML. Puede ver el mapeo que este codificador ha aprendido usando el clases_ atributo ("<1H OCEAN" se asigna a 0, "INLAND" se asigna a 1, etc.):

```
>>> impresión ( codificador . clases_ )
['<1H OCEAN' 'INLAND' 'ISLAND' 'NEAR BAY' 'NEAR OCEAN']
```

Un problema con esta representación es que los algoritmos ML supondrán que dos valores cercanos son más similares que dos valores distantes. Obviamente, este no es el caso (por ejemplo, las categorías 0 y 4 son más similares que las categorías 0 y 1). Para solucionar este problema, una solución común es crear un atributo binario por categoría: un atributo igual a 1 cuando la categoría es "<1H OCEAN" (y 0 en caso contrario), otro atributo igual a 1 cuando la categoría es "INLAND" ( y 0 en caso contrario), y así sucesivamente. Se llama *codificación one-hot,* porque solo un atributo será igual a 1 (caliente), mientras que los otros serán 0 (frío).

Scikit-Learn proporciona una OneHotEncoder codificador para convertir valores categóricos enteros en vectores one-hot. Codifiquemos las categorías como vectores one-hot. Tenga en cuenta que fit_transform () espera una matriz 2D, pero Housing_cat_encoded es una matriz 1D, por lo que Necesito remodelarlo: [18]

```
>>> de sklearn.preprocessing importar OneHotEncoder
>>> codificador = OneHotEncoder ()
>>> housing_cat_1hot = codificador . fit_transform ( Housing_cat_encoded . remodelar ( - 1 , 1 ))
>>> housing_cat_1hot
<16513x5 matriz dispersa de tipo '<class' numpy.float64 '>'
        con 16513 elementos almacenados en formato Compressed Sparse Row>
```

Note que la salida es un SciPy *matriz dispersa,* en lugar de una matriz NumPy. Esto es muy útil cuando tiene atributos categóricos con miles de categorías. Después de una codificación en caliente, obtenemos una matriz con miles de columnas, y la matriz está llena de ceros, excepto uno por fila. Utilizar toneladas de memoria principalmente para almacenar ceros sería un desperdicio, por lo que una matriz dispersa solo almacena la ubicación de los elementos distintos de cero. Puede usarlo principalmente como una matriz 2D normal, [19] pero si realmente desea convertirlo en una matriz NumPy (densa), simplemente llame al toarray () método:

```
>>> housing_cat_1hot . toarray ()
matriz ([[0., 1., 0., 0., 0.],
        [0., 1., 0., 0., 0.], [0., 0., 0., 0., 1.],

        . . . ,
        [0.,    1.,    0.,    0.,    0.],
```

18 NumPy remodelar () La función permite que una dimensión sea –1, lo que significa "sin especificar": el valor se infiere de la longitud de la matriz y las dimensiones restantes.

19 Consulte la documentación de SciPy para obtener más detalles.

```
       [1., 0., 0., 0., 0.], [0., 0., 0., 1., 0.]])
```

Podemos aplicar ambas transformaciones (de categorías de texto a categorías de números enteros, luego de categorías de números enteros a vectores one-hot) en una sola toma usando el LabelBinarizer clase:

```
>>> de sklearn.preprocessing importar LabelBinarizer
>>> codificador = LabelBinarizer ()
>>> housing_cat_1hot = codificador . fit_transform ( vivienda_gato )
>>> housing_cat_1hot
matriz ([[0, 1, 0, 0, 0],
       [0, 1, 0, 0, 0], [0, 0, 0, 0,
       1],
       . . . ,
       [0, 1, 0, 0, 0], [1, 0, 0, 0,
       0], [0, 0, 0, 1, 0]])
```

Tenga en cuenta que esto devuelve una matriz NumPy densa de forma predeterminada. Puede obtener una matriz dispersa en su lugar pasando sparse_output = Verdadero al LabelBinarizer constructor.

## Transformadores personalizados

Aunque Scikit-Learn proporciona muchos transformadores útiles, deberá escribir los suyos propios para tareas como operaciones de limpieza personalizadas o combinación de atributos específicos. Querrá que su transformador funcione a la perfección con las funciones de Scikit-Learn (como pipelines), y dado que Scikit-Learn se basa en la escritura de pato (no en la herencia), todo lo que necesita es crear una clase e implementar tres métodos: ajuste()
(regresando self), transform (), y fit_transform (). Puede obtener el último gratis simplemente agregando TransformerMixin como clase base. Además, si agrega BaseEstima tor como clase base (y evitar * argumentos y ** kargs en su constructor) obtendrá dos métodos adicionales ( get_params () y set_params ()) que será útil para el ajuste automático de hiperparámetros. Por ejemplo, aquí hay una pequeña clase de transformador que agrega los atributos combinados que discutimos anteriormente:

```
de sklearn.base importar BaseEstimator , TransformerMixin

rooms_ix , dormitorios_ix , población_ix , hogar_ix = 3 , 4 , 5 , 6

clase CombinedAttributesAdder ( BaseEstimator , TransformerMixin ):
    def __en eso__ ( yo , add_bedrooms_per_room = Cierto ): # sin * args o ** kargs
        yo . add_bedrooms_per_room = add_bedrooms_per_room
    def ajuste ( yo , X , y = Ninguna ):
        regreso yo # nada más que hacer
    def transformar ( yo , X , y = Ninguna ):
        rooms_per_household = X [:, rooms_ix ] / X [:, hogar_ix ]
        población_por_familia = X [:, población_ix ] / X [:, hogar_ix ]
        Si yo . add_bedrooms_per_room :
            dormitorios_por_habitación = X [:, dormitorios_ix ] / X [:, rooms_ix ]
```

```
            regreso notario público . C_ [ X , rooms_per_household , población_por_familia ,
                                           dormitorios_por_habitación ]
        más :
            regreso notario público . C_ [ X , rooms_per_household , población_por_familia ]

attr_adder = CombinedAttributesAdder ( add_bedrooms_per_room = Falso )
housing_extra_attribs = attr_adder . transformar ( alojamiento . valores )
```

En este ejemplo, el transformador tiene un hiperparámetro, add_bedrooms_per_room, ajustado a Cierto por defecto (a menudo es útil proporcionar valores predeterminados razonables). Este hiperparámetro le permitirá averiguar fácilmente si agregar este atributo ayuda a los algoritmos de Machine Learning o no. De manera más general, puede agregar un hiperparámetro para controlar cualquier paso de preparación de datos del que no esté 100% seguro. Cuanto más automatice estos pasos de preparación de datos, más combinaciones podrá probar automáticamente, por lo que es mucho más probable que encuentre una gran combinación (y le ahorrará mucho tiempo).

## Escala de características

Una de las transformaciones más importantes que debe aplicar a sus datos es *escala de características*. Con pocas excepciones, los algoritmos de aprendizaje automático no funcionan bien cuando los atributos numéricos de entrada tienen escalas muy diferentes. Este es el caso de los datos de vivienda: el número total de habitaciones varía de aproximadamente 6 a 39,320, mientras que los ingresos medios solo varían de 0 a 15. Tenga en cuenta que generalmente no es necesario escalar los valores objetivo.

Hay dos formas comunes de conseguir que todos los atributos tengan la misma escala: *escala mínima-máxima* y *Estandarización*.

Escalado mínimo-máximo (mucha gente llama a esto *normalización)* es bastante simple: los valores se cambian y se reescalan de modo que terminen oscilando entre 0 y 1. Esto lo hacemos restando el valor mínimo y dividiéndolo por el máximo menos el mínimo. Scikit-Learn proporciona un transformador llamado MinMaxScaler para esto. Tiene un feature_range hiperparámetro que le permite cambiar el rango si no quiere 0–1 por alguna razón.

La estandarización es bastante diferente: primero resta el valor medio (por lo que los valores estandarizados siempre tienen una media cero), y luego divide por la varianza para que la distribución resultante tenga varianza unitaria. A diferencia del escalado mínimo-máximo, la estandarización no limita los valores a un rango específico, lo que puede ser un problema para algunos algoritmos (por ejemplo, las redes neuronales a menudo esperan un valor de entrada que varía de 0 a 1). Sin embargo, la estandarización se ve mucho menos afectada por los valores atípicos. Por ejemplo, suponga que un distrito tiene un ingreso medio igual a 100 (por error). El escalado mínimo-máximo aplastaría todos los demás valores de 0-15 a 0-0,15, mientras que la estandarización no se vería muy afectada. Scikit-Learn proporciona un transformador llamado Escalador estándar para la estandarización.

Al igual que con todas las transformaciones, es importante ajustar los escaladores solo a los datos de entrenamiento, no al conjunto de datos completo (incluido el conjunto de prueba). Solo entonces puede usarlos para transformar el conjunto de entrenamiento y el conjunto de prueba (y nuevos datos).

## Canalizaciones de transformación

Como puede ver, hay muchos pasos de transformación de datos que deben ejecutarse en el orden correcto. Afortunadamente, Scikit-Learn proporciona Tubería class para ayudar con tales secuencias de transformaciones. Aquí hay una pequeña canalización para los atributos numéricos:

```
de sklearn.pipeline importar Tubería
de sklearn.preprocessing importar Escalador estándar

num_pipeline = Tubería ([
        ( 'imputador' , Imputer ( estrategia = "mediana" )),
        ( 'attribs_adder' , CombinedAttributesAdder ()),
        ( 'std_scaler' , Escalador estándar ()),
    ])

Housing_num_tr = num_pipeline . fit_transform ( número_vivienda )
```

los Tubería constructor toma una lista de pares de nombre / estimador que definen una secuencia de pasos. Todos menos el último estimador deben ser transformadores (es decir, deben tener un fit_transform () método). Los nombres pueden ser los que quieras.

Cuando llamas a la tubería ajuste() método, llama fit_transform () secuencialmente en todos los transformadores, pasando la salida de cada llamada como parámetro a la siguiente llamada, hasta que llega al estimador final, para lo cual solo llama al ajuste() método.

La tubería expone los mismos métodos que el estimador final. En este ejemplo, el último estimador es un Escalador estándar, que es un transformador, por lo que la tubería tiene un forma trans () método que aplica todas las transformaciones a los datos en secuencia (también tiene un fit_transform método que podríamos haber usado en lugar de llamar ajuste() y entonces transformar()).

Ahora tiene una canalización para valores numéricos y también necesita aplicar la LabelBi narizador sobre los valores categóricos: ¿cómo se pueden unir estas transformaciones en una única tubería? Scikit-Learn proporciona una FeatureUnion clase para esto. Le da una lista de transformadores (que pueden ser conductos de transformadores completos), y cuando su transformar() se llama método, ejecuta cada transformador transformar() método en paralelo, espera su salida, y luego los concatena y devuelve el resultado (y por supuesto llamando a su ajuste() El método llama a todos los transformadores ajuste() método). Una canalización completa que maneja atributos tanto numéricos como categóricos puede verse así:

```
de sklearn.pipeline importar FeatureUnion

num_attribs = lista ( número_vivienda )
cat_attribs = [ "ocean_proximity" ]

num_pipeline = Tubería ([
        ( 'selector' , DataFrameSelector ( num_attribs )),
        ( 'imputador' , Imputer ( estrategia = "mediana" )),
        ( 'attribs_adder' , CombinedAttributesAdder ()),
        ( 'std_scaler' , Escalador estándar ()),
    ])

cat_pipeline = Tubería ([
        ( 'selector' , DataFrameSelector ( cat_attribs )),
        ( 'label_binarizer' , LabelBinarizer ()),
    ])

full_pipeline = FeatureUnion ( lista_transformadores = [
        ( "num_pipeline" , num_pipeline ),
        ( "cat_pipeline" , cat_pipeline ),
    ])
```

Y puede ejecutar toda la canalización simplemente:

```
>>> vivienda_preparada = full_pipeline . fit_transform ( alojamiento )
>>> vivienda_preparada
matriz ([[0.73225807, -0.67331551, 0.58426443, ...,                0.                ,
          0.                , 0.                ],
         [-0,99102923, 1,63234656, -0,92655887, ...,                0.                ,
          0.                , 0.                ],
         [...]
>>> vivienda_preparada . forma
(16513, 17)
```

Cada subtubo comienza con un transformador selector: simplemente transforma los datos seleccionando los atributos deseados (numéricos o categóricos), descartando el resto y convirtiendo el DataFrame resultante en una matriz NumPy. No hay nada en Scikit-Learn para manejar Pandas DataFrames,[20] así que necesitamos escribir un transformador personalizado simple para esta tarea:

```
de sklearn.base importar BaseEstimator , TransformerMixin

clase DataFrameSelector ( BaseEstimator , TransformerMixin ):
    def __en eso__ ( yo , nombres_atributos ):
        yo . nombres_atributos = nombres_atributos
    def ajuste ( yo , X , y = Ninguna ):
        regreso yo
```

20 Pero consulte Pull Request # 3886, que puede introducir una ColumnTransformer clase que hace el atributo específico transformaciones fáciles. También podrías correr pip3 instalar sklearn-pandas conseguir un DataFrameMapper clase con un objetivo similar.

```
    def transformar ( yo , X ):
        regreso X [ yo . nombres_atributos ] . valores
```

## Seleccionar y entrenar un modelo

¡Al final! Enmarcó el problema, obtuvo los datos y los exploró, tomó muestras de un conjunto de entrenamiento y un conjunto de prueba, y escribió canales de transformación para limpiar y preparar sus datos para los algoritmos de aprendizaje automático automáticamente. Ahora está listo para seleccionar y entrenar un modelo de aprendizaje automático.

### Capacitación y evaluación en el conjunto de capacitación

La buena noticia es que gracias a todos estos pasos anteriores, las cosas ahora van a ser mucho más simples de lo que piensas. Primero entrenemos un modelo de regresión lineal, como hicimos en el capítulo anterior:

```
de sklearn.linear_model importar Regresión lineal

lin_reg = Regresión lineal ()
lin_reg . ajuste ( vivienda_preparada , viviendas_etiquetas )
```

¡Hecho! Ahora tiene un modelo de regresión lineal en funcionamiento. Probémoslo en algunas instancias del conjunto de entrenamiento:

```
>>> algunos datos = alojamiento . iloc [: 5 ]
>>> some_labels = viviendas_etiquetas . iloc [: 5 ]
>>> some_data_prepared = full_pipeline . transformar ( algunos datos )
>>> impresión ( " Predicciones: \ t " , lin_reg . predecir ( some_data_prepared ))
Predicciones: [303104. 44800. 308928. 294208. 368704.]
>>> impresión ( " Etiquetas:\ t \ t " , lista ( some_labels ))
Etiquetas: [359400.0, 69700.0, 302100.0, 301300.0, 351900.0]
```

Funciona, aunque las predicciones no son exactamente precisas (por ejemplo, ¡la segunda predicción tiene una diferencia de más del 50%!). Midamos el RMSE de este modelo de regresión en todo el conjunto de entrenamiento usando Scikit-Learn's error medio cuadrado función:

```
>>> de sklearn.metrics importar error medio cuadrado
>>> predicciones_de_vivienda = lin_reg . predecir ( vivienda_preparada )
>>> lin_mse = error medio cuadrado ( viviendas_etiquetas , predicciones_de_vivienda )
>>> lin_rmse = notario público . sqrt ( lin_mse )
>>> lin_rmse
68628.413493824875
```

Bien, esto es mejor que nada, pero claramente no es una gran puntuación: la mayoría de los distritos ' median_housing_values oscilan entre $ 120,000 y $ 265,000, por lo que un error de predicción típico de $ 68,628 no es muy satisfactorio. Este es un ejemplo de un modelo que no se ajusta a los datos de entrenamiento. Cuando esto sucede, puede significar que las características no brindan suficiente información para hacer buenas predicciones o que el modelo no es lo suficientemente poderoso. Como vimos en el capítulo anterior, las principales formas de corregir el desajuste son

seleccione un modelo más potente para alimentar el algoritmo de entrenamiento con mejores funciones o para reducir las limitaciones del modelo. Este modelo no está regularizado, por lo que se descarta la última opción. Podría intentar agregar más características (por ejemplo, el registro de la población), pero primero intentemos un modelo más complejo para ver cómo funciona.

Entrenemos un DecisionTreeRegressor. Este es un modelo poderoso, capaz de encontrar relaciones complejas no lineales en los datos (los árboles de decisión se presentan con más detalle en Capítulo 6 ). El código debería parecer familiar ahora:

```
de sklearn.tree importar DecisiónÁrbolRegresor

tree_reg = DecisiónÁrbolRegresor ()
tree_reg . ajuste ( vivienda_preparada , viviendas_etiquetas )
```

Ahora que el modelo está entrenado, evaluémoslo en el conjunto de entrenamiento:

```
>>> predicciones_de_vivienda = tree_reg . predecir ( vivienda_preparada )
>>> tree_mse = error medio cuadrado ( viviendas_etiquetas , predicciones_de_vivienda )
>>> tree_rmse = notario público . sqrt ( tree_mse )
>>> tree_rmse
0.0
```

¿¡Esperar lo!? ¿Ningún error en absoluto? ¿Podría este modelo ser absolutamente perfecto? Por supuesto, es mucho más probable que el modelo se haya ajustado demasiado a los datos. ¿Como puedes estar seguro? Como vimos anteriormente, no desea tocar el conjunto de prueba hasta que esté listo para lanzar un modelo en el que esté seguro, por lo que debe usar parte del conjunto de entrenamiento para la capacitación y parte para la validación del modelo.

## Mejor evaluación mediante validación cruzada

Una forma de evaluar el modelo de árbol de decisión sería utilizar la train_test_split función para dividir el conjunto de entrenamiento en un conjunto de entrenamiento más pequeño y un conjunto de validación, luego entrene sus modelos con el conjunto de entrenamiento más pequeño y evalúelos frente al conjunto de validación. Es un poco de trabajo, pero nada demasiado difícil y funcionaría bastante bien.

Una gran alternativa es utilizar Scikit-Learn's *validación cruzada* característica. El siguiente código realiza *Validación cruzada de K-fold:* divide aleatoriamente el conjunto de entrenamiento en 10 subconjuntos distintos llamados *pliegues* luego entrena y evalúa el modelo de árbol de decisión 10 veces, seleccionando un pliegue diferente para evaluarlo cada vez y entrenando en los otros 9 pliegues. El resultado es una matriz que contiene las 10 puntuaciones de evaluación:

```
de sklearn.model_selection importar puntuaciones de cross_val_score = cross_val_score ( tree_reg , vivienda_preparada
, viviendas_etiquetas ,
                                         puntuación = "neg_mean_squared_error" , CV = 10 )
rmse_scores = notario público . sqrt ( - puntuaciones )
```

Las características de validación cruzada de Scikit-Learn esperan una función de utilidad (cuanto mayor es mejor) en lugar de una función de costo (menor es mejor), por lo que la función de puntuación es en realidad lo opuesto al MSE (es decir, un valor negativo), que es por qué se calcula el código anterior - puntuaciones
antes de calcular la raíz cuadrada.

Veamos los resultados:

```
>>> def display_scores ( puntuaciones ):
...         impresión ( " Puntuaciones:" , puntuaciones )
...         impresión ( " Media:" , puntuaciones . media ())
...         impresión ( " Desviación Estándar:" , puntuaciones . std ())
...
>>> display_scores ( tree_rmse_scores )
Puntuaciones: [74678.4916885          64766.2398337          69632.86942005 69166.67693232
                71486.76507766 73321.65695983 71860.04741226 71086.32691692
                76934.2726093          69060.93319262]
Media: 71199.4280043
Desviación estándar: 3202.70522793
```

Ahora, el árbol de decisiones no se ve tan bien como antes. De hecho, parece funcionar peor que el modelo de regresión lineal. Tenga en cuenta que la validación cruzada le permite obtener no solo una estimación del rendimiento de su modelo, sino también una medida de la precisión de esta estimación (es decir, su desviación estándar). El árbol de decisiones tiene una puntuación de aproximadamente 71.200, generalmente ± 3.200. No tendría esta información si solo usara un conjunto de validación. Pero la validación cruzada tiene el costo de entrenar el modelo varias veces, por lo que no siempre es posible.

Calculemos las mismas puntuaciones para el modelo de regresión lineal solo para estar seguros:

```
>>> lin_scores = cross_val_score ( lin_reg , vivienda_preparada , viviendas_etiquetas ,
...                                 puntuación = "neg_mean_squared_error" , CV = 10 )
...
>>> lin_rmse_scores = notario público . sqrt ( - lin_scores )
>>> display_scores ( lin_rmse_scores )
Puntuaciones: [70423.5893262          65804.84913139          66620.84314068          72510.11362141
                66414.74423281 71958.89083606          67624.90198297          67825.36117664
                72512.36533141 68028.11688067]
Media: 68972,377566
Desviación estándar: 2493.98819069
```

Así es: el modelo de árbol de decisión se sobreajusta tan mal que funciona peor que el modelo de regresión lineal.

Probemos ahora un último modelo: el RandomForestRegressor. Como veremos en Capítulo 7 Los bosques aleatorios funcionan entrenando muchos árboles de decisión en subconjuntos aleatorios de las características y luego promediando sus predicciones. La construcción de un modelo sobre muchos otros modelos se llama *Aprendizaje conjunto,* y a menudo es una excelente manera de impulsar los algoritmos de aprendizaje automático aún más. Omitiremos la mayor parte del código, ya que es esencialmente el mismo que para los otros modelos:

```
>>> de sklearn.ensemble importar AleatorioBosqueRegresor
>>> forest_reg = AleatorioBosqueRegresor ()
>>> forest_reg . ajuste ( vivienda_preparada , viviendas_etiquetas )
>>> [ ... ]
>>> forest_rmse
22542.396440343684
>>> display_scores ( forest_rmse_scores )
Puntuaciones: [53789.2879722 50256.19806622 52521.55342602 53237.44937943
 52428.82176158 55854.61222549 52158.02291609 50093.66125649
 53240.80406125 52761.50852822]
Media: 52634.1919593
Desviación estándar: 1576.20472269
```

Vaya, esto es mucho mejor: los bosques aleatorios parecen muy prometedores. Sin embargo, tenga en cuenta que la puntuación en el conjunto de entrenamiento sigue siendo mucho más baja que en los conjuntos de validación, lo que significa que el modelo todavía está sobreajustando el conjunto de entrenamiento. Las posibles soluciones para el sobreajuste son simplificar el modelo, restringirlo (es decir, regularizarlo) u obtener muchos más datos de entrenamiento. Sin embargo, antes de profundizar mucho más en Random Forests, debería probar muchos otros modelos de varias categorías de algoritmos de aprendizaje automático (varias máquinas de vectores de soporte con diferentes núcleos, posiblemente una red neuronal, etc.), sin perder demasiado tiempo ajustar los hiperparámetros. El objetivo es preseleccionar algunos modelos prometedores (de dos a cinco).

Debe guardar todos los modelos con los que experimenta, para poder volver fácilmente a cualquier modelo que desee. Asegúrese de guardar tanto los hiperparámetros como los parámetros entrenados, así como las puntuaciones de validación cruzada y quizás también las predicciones reales. Esto le permitirá comparar fácilmente las puntuaciones entre los tipos de modelos y comparar los tipos de errores que cometen. Puede guardar fácilmente modelos de Scikit-Learn usando Python pepinillo módulo, o usando

sklearn.externals.joblib, que es más eficiente en la serialización de grandes matrices NumPy:

```
de sklearn.externals importar joblib

joblib . tugurio ( mi modelo , "my_model.pkl" )
# y después...
my_model_loaded = joblib . carga ( "my_model.pkl" )
```

# Ajuste su modelo

Supongamos que ahora tiene una lista corta de modelos prometedores. Ahora necesita ajustarlos. Veamos algunas formas en las que puede hacerlo.

## Búsqueda de cuadrícula

Una forma de hacerlo sería jugar con los hiperparámetros manualmente, hasta que encuentre una gran combinación de valores de hiperparámetros. Este sería un trabajo muy tedioso y es posible que no tenga tiempo para explorar muchas combinaciones.

En su lugar, debería obtener Scikit-Learn's GridSearchCV para buscarte. Todo lo que necesita hacer es decirle con qué hiperparámetros desea que experimente y qué valores probar, y evaluará todas las combinaciones posibles de valores de hiperparámetros, utilizando la validación cruzada. Por ejemplo, el siguiente código busca la mejor combinación de valores de hiperparámetros para el RandomForestRegressor:

```
de sklearn.model_selection importar GridSearchCV

param_grid = [
        { 'n_estimators' : [ 3 , 10 , 30 ], 'max_features' : [ 2 , 4 , 6 , 8 ]}, { 'oreja' : [ Falso ], 'n_estimators' : [ 3 , 10 ], 'max_features' : [ 2 , 3
        , 4 ]},
    ]

forest_reg = AleatorioBosqueRegresor ()

grid_search = GridSearchCV ( forest_reg , param_grid , CV = 5 ,
                                            puntuación = 'neg_mean_squared_error' )

grid_search . ajuste ( vivienda_preparada , viviendas_etiquetas )
```

Cuando no tiene idea de qué valor debería tener un hiperparámetro, un enfoque simple es probar potencias consecutivas de 10 (o un número menor si desea una búsqueda más detallada, como se muestra en este ejemplo con el n_estimators hiperparámetro).

Esta param_grid le dice a Scikit-Learn que primero evalúe todas las combinaciones 3 × 4 = 12 de n_estimators y max_features valores de hiperparámetros especificados en el primer dictar (no se preocupe por el significado de estos hiperparámetros por ahora; se explicarán en Capítulo 7 ), luego intente todas las combinaciones 2 × 3 = 6 de valores de hiperparámetros en el segundo dictar pero esta vez con el oreja hiperparámetro establecido en Falso en vez de Cierto ( que es el valor predeterminado para este hiperparámetro).

Con todo, la búsqueda de la cuadrícula explorará 12 + 6 = 18 combinaciones de RandomForestRe gressor valores de hiperparámetros, y entrenará cada modelo cinco veces (ya que estamos usando una validación cruzada de cinco veces). En otras palabras, en total, ¡habrá 18 × 5 = 90 rondas de entrenamiento! Puede llevar bastante tiempo, pero una vez hecho, puede obtener la mejor combinación de parámetros como esta:

```
>>>  grid_search . best_params_
{'max_features': 6, 'n_estimators': 30}
```

Dado que 30 es el valor máximo de n_estimators que se evaluó, probablemente también debería evaluar valores más altos, ya que la puntuación puede seguir mejorando.

También puede obtener el mejor estimador directamente:

```
>>> grid_search . best_estimator_
RandomForestRegressor (bootstrap = True, criterio = 'mse', max_depth = Ninguno,
                       max_features = 6, max_leaf_nodes = Ninguno, min_samples_leaf = 1,
                       min_samples_split = 2, min_weight_fraction_leaf = 0.0,
                       n_estimators = 30, n_jobs = 1, oob_score = False, random_state = None, verbose = 0,
                       warm_start = False)
```

Si GridSearchCV se inicializa con reajustar = Verdadero ( que es el predeterminado), una vez que encuentra el mejor estimador mediante validación cruzada, lo vuelve a entrenar en todo el conjunto de entrenamiento. Esto suele ser una buena idea, ya que alimentarlo con más datos probablemente mejorará su rendimiento.

Y, por supuesto, las puntuaciones de la evaluación también están disponibles:

```
>>> cvres = grid_search . cv_results_
. . . para puntuación media , params en Código Postal ( cvres [ "mean_test_score" ], cvres [ "params" ]):
. . . impresión ( notario público . sqrt ( - puntuación media ), params )
. . .
64912.0351358 {'max_features': 2, 'n_estimators': 3}
55535.2786524 {'max_features': 2, 'n_estimators': 10}
52940.2696165 {'max_features': 2, 'n_estimators': 30}
60384.0908354 {'max_features': 4, 'n_estimators': 3}
52709.9199934 {'max_features': 4, 'n_estimators': 10}
50503.5985321 {'max_features': 4, 'n_estimators': 30}
59058.1153485 {'max_features': 6, 'n_estimators': 3}
52172.0292957 {'max_features': 6, 'n_estimators': 10}
49958.9555932 {'max_features': 6, 'n_estimators': 30}
59122.260006 {'max_features': 8, 'n_estimators': 3}
52441.5896087 {'max_features': 8, 'n_estimators': 10}
50041.4899416 {'max_features': 8, 'n_estimators': 30}
62371.1221202 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
54572.2557534 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
59634.0533132 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
52456.0883904 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
58825.665239 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
52012.9945396 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

En este ejemplo, obtenemos la mejor solución configurando el max_features hiperparámetro a 6, y el n_estimators hiperparámetro a 30. El puntaje RMSE para esta combinación es 49,959, que es un poco mejor que el puntaje que obtuvo anteriormente con el

valores de hiperparámetros predeterminados (que era 52,634). ¡Enhorabuena, ha perfeccionado con éxito su mejor modelo!

No olvide que puede tratar algunos de los pasos de preparación de datos como hiperparámetros. Por ejemplo, la búsqueda en la cuadrícula encontrará automáticamente si agregar o no una característica de la que no estaba seguro (por ejemplo, usando el add_bedrooms_per_room hiperparámetro de tu CombinedAttributesAdder transformador). De manera similar, se puede usar para encontrar automáticamente la mejor manera de manejar valores atípicos, características faltantes, selección de características y más.

## Búsqueda aleatoria

El enfoque de búsqueda de cuadrícula está bien cuando está explorando relativamente pocas combinaciones, como en el ejemplo anterior, pero cuando el hiperparámetro *espacio de búsqueda* es grande, a menudo es preferible usar RandomizedSearchCV en lugar. Esta clase se puede utilizar de la misma forma que la GridSearchCV clase, pero en lugar de probar todas las combinaciones posibles, evalúa un número dado de combinaciones aleatorias seleccionando un valor aleatorio para cada hiperparámetro en cada iteración. Este enfoque tiene dos beneficios principales:

- Si deja que la búsqueda aleatoria se ejecute, digamos, 1,000 iteraciones, este enfoque explorará 1,000 valores diferentes para cada hiperparámetro (en lugar de solo unos pocos valores por hiperparámetro con el enfoque de búsqueda de cuadrícula).
- Tiene más control sobre el presupuesto de cálculo que desea asignar a la búsqueda de hiperparámetros, simplemente configurando el número de iteraciones.

## Métodos de conjunto

Otra forma de ajustar su sistema es tratar de combinar los modelos que funcionan mejor. El grupo (o "conjunto") a menudo se desempeñará mejor que el mejor modelo individual (al igual que los bosques aleatorios se desempeñan mejor que los árboles de decisión individuales en los que se basan), especialmente si los modelos individuales cometen tipos de errores muy diferentes. Cubriremos este tema con más detalle en Capítulo 7 .

## Analizar los mejores modelos y sus errores

A menudo obtendrá buenos conocimientos sobre el problema al inspeccionar los mejores modelos. Por ejemplo, el AleatorioBosqueRegresor puede indicar la importancia relativa de cada atributo para realizar predicciones precisas:

```
>>> feature_importances = grid_search . best_estimator_ . feature_importances_
>>> feature_importances
matriz ([7.14156423e-02, 6.76139189e-02, 4.44260894e-02,
```

```
1.66308583e-02,  1.66076861e-02,  1.82402545e-02,
1.63458761e-02,  3.26497987e-01,  6.04365775e-02,
1.13055290e-01,  7.79324766e-02,  1.12166442e-02,
1.53344918e-01,  8.41308969e-05,  2.68483884e-03,
3.46681181e-03])
```

Vamos a mostrar estas puntuaciones de importancia junto a sus correspondientes nombres de atributos:

```
>>> extra_attribs = [ "rooms_per_hhold" , "pop_per_hhold" , "dormitorios_por_habitación" ]
>>> cat_one_hot_attribs = lista ( codificador . clases_ )
>>> atributos = num_attribs + extra_attribs + cat_one_hot_attribs
>>> ordenado ( Código Postal ( feature_importances , atributos ), marcha atrás = Cierto )
[(0.32649798665134971, 'ingresos_medios'),
 (0.15334491760305854, 'INTERIOR'),
 (0.11305529021187399, 'pop_per_hhold'),
 (0.0779324766254475, 'habitaciones_por_habitación'),
 (0.071415642259275158, 'longitud'),
 (0.067613918945568688, 'latitud'),
 (0.060436577499703222, 'rooms_per_hhold'),
 (0.04442608939578685, 'edad_mediana_de_vivienda'),
 (0.018240254462909437, 'población'),
 (0.01663085833886218, 'total_rooms'),
 (0.016607686091288865, 'total_bedrooms'),
 (0.016345876147580776, 'hogares'),
 (0.011216644219017424, '<1H OCÉANO'),
 (0.0034668118081117387, 'CERCA DEL OCÉANO'),
 (0.0026848388432755429, 'CERCA DE LA BAHÍA'),
 (8.4130896890070617e-05, 'ISLA')]
```

Con esta información, es posible que desee intentar eliminar algunas de las funciones menos útiles (por ejemplo, aparentemente solo una oceano_proximidad La categoría es realmente útil, por lo que podría intentar eliminar las otras).

También debe observar los errores específicos que comete su sistema, luego tratar de comprender por qué los comete y qué podría solucionar el problema (agregar funciones adicionales o, por el contrario, deshacerse de los que no son informativos, eliminar valores atípicos, etc. .).

## Evalúe su sistema en el equipo de prueba

Después de ajustar sus modelos por un tiempo, eventualmente tendrá un sistema que funciona suficientemente bien. Ahora es el momento de evaluar el modelo final en el conjunto de prueba. No hay nada especial en este proceso; simplemente obtenga los predictores y las etiquetas de su conjunto de prueba, ejecute su full_pipeline para transformar los datos (llamar transformar(), *no*
fit_transform ()!), y evaluar el modelo final en el conjunto de prueba:

```
modelo_final = grid_search . best_estimator_

X_test = strat_test_set . soltar ( "valor_median_house" , eje = 1 )
y_test = strat_test_set [ "valor_median_house" ] . Copiar ()

X_test_prepared = full_pipeline . transformar ( X_test )
```

```
predicciones_finales = modelo_final . predecir ( X_test_prepared )

final_mse = error medio cuadrado ( y_test , predicciones_finales )
final_rmse = notario público . sqrt ( final_mse )              # => se evalúa como 48,209.6
```

Por lo general, el rendimiento será un poco peor que el que midió con la validación cruzada si realizó muchos ajustes de hiperparámetros (porque su sistema termina ajustado para funcionar bien en los datos de validación y probablemente no funcionará tan bien en conjuntos de datos desconocidos ). No es el caso en este ejemplo, pero cuando esto sucede, debe resistir la tentación de modificar los hiperparámetros para que los números se vean bien en el conjunto de prueba; Es poco probable que las mejoras se generalicen a nuevos datos.

Ahora viene la fase de prelanzamiento del proyecto: debe presentar su solución (resaltar lo que ha aprendido, lo que funcionó y lo que no, qué suposiciones se hicieron y cuáles son las limitaciones de su sistema), documentar todo y crear presentaciones agradables con visualizaciones claras y declaraciones fáciles de recordar (por ejemplo, “el ingreso medio es el predictor número uno de los precios de la vivienda”).

## Inicie, supervise y mantenga su sistema

Perfecto, ¡tienes aprobación para lanzar! Necesita preparar su solución para la producción, en particular conectando las fuentes de datos de entrada de producción a su sistema y escribiendo pruebas.

También necesita escribir un código de monitoreo para verificar el rendimiento en vivo de su sistema a intervalos regulares y activar alertas cuando caiga. Esto es importante para detectar no solo roturas repentinas, sino también degradación del rendimiento. Esto es bastante común porque los modelos tienden a "pudrirse" a medida que los datos evolucionan con el tiempo, a menos que los modelos se entrenan regularmente con datos nuevos.

La evaluación del rendimiento de su sistema requerirá tomar muestras de las predicciones del sistema y evaluarlas. Esto generalmente requerirá un análisis humano. Estos analistas pueden ser expertos de campo o trabajadores de una plataforma de crowdsourcing (como Amazon Mechanical Turk o CrowdFlower). De cualquier manera, necesita conectar la tubería de evaluación humana a su sistema.

También debe asegurarse de evaluar la calidad de los datos de entrada del sistema. A veces, el rendimiento se degradará levemente debido a una señal de mala calidad (por ejemplo, un sensor que no funciona correctamente que envía valores aleatorios o la salida de otro equipo se vuelve obsoleta), pero puede pasar un tiempo antes de que el rendimiento de su sistema se degrade lo suficiente como para activar una alerta. Si monitorea las entradas de su sistema, puede detectar esto antes. Monitorear las entradas es particularmente importante para los sistemas de aprendizaje en línea.

Por último, generalmente querrá entrenar sus modelos de forma regular utilizando datos nuevos. Debe automatizar este proceso tanto como sea posible. Si no lo hace, es muy

es probable que actualice su modelo solo cada seis meses (en el mejor de los casos), y el rendimiento de su sistema puede fluctuar severamente con el tiempo. Si su sistema es un sistema de aprendizaje en línea, debe asegurarse de guardar instantáneas de su estado a intervalos regulares para que pueda volver fácilmente a un estado de trabajo anterior.

## ¡Pruébalo!

Con suerte, este capítulo le dio una buena idea de cómo es un proyecto de aprendizaje automático y le mostró algunas de las herramientas que puede utilizar para entrenar un gran sistema. Como puede ver, gran parte del trabajo está en el paso de preparación de datos, la creación de herramientas de monitoreo, la configuración de canales de evaluación humana y la automatización de la capacitación periódica del modelo. Los algoritmos de aprendizaje automático también son importantes, por supuesto, pero probablemente sea preferible sentirse cómodo con el proceso general y conocer bien tres o cuatro algoritmos en lugar de dedicar todo el tiempo a explorar algoritmos avanzados y no dedicarle suficiente tiempo al proceso general. .

Entonces, si aún no lo ha hecho, ahora es un buen momento para tomar una computadora portátil, seleccionar un conjunto de datos que le interese e intentar pasar por todo el proceso de A a Z. Un buen lugar para comenzar es en un sitio web de competencia como *http://kaggle.com/ :* tendrás un conjunto de datos con el que jugar, un objetivo claro y personas con las que compartir la experiencia.

## Ejercicios

Usando el conjunto de datos de vivienda de este capítulo:

1. Pruebe un regresor de máquina de vectores de soporte ( `sklearn.svm.SVR`), con varios hiper-parámetros como `kernel = "lineal"` ( con varios valores para el `C` hiperparámetro) o `kernel = "rbf"` ( con varios valores para el `C` y `gama` hiperparámetros). No se preocupe por lo que significan estos hiperparámetros por ahora. ¿Cómo funciona el mejor `SVR` predictor realizar?
2. Intente reemplazar `GridSearchCV` con `RandomizedSearchCV`.
3. Intente agregar un transformador en la tubería de preparación para seleccionar solo los atributos más importantes.
4. Intente crear una única canalización que realice la preparación completa de los datos más la predicción final.
5. Explore automáticamente algunas opciones de preparación utilizando `GridSearchCV`.

Las soluciones para estos ejercicios están disponibles en los cuadernos de Jupyter en línea en *https: // github.com/ageron/handson-ml* .

CAPÍTULO 3

# Clasificación

En Capítulo 1 mencionamos que las tareas de aprendizaje supervisado más comunes son regresión (predicción de valores) y clasificación (predicción de clases). En Capitulo 2 exploramos una tarea de regresión, prediciendo el valor de la vivienda, utilizando varios algoritmos como Regresión lineal, Árboles de decisión y Bosques aleatorios (que se explicarán con más detalle en capítulos posteriores). Ahora centraremos nuestra atención en los sistemas de clasificación.

## MNIST

En este capítulo, usaremos el conjunto de datos del MNIST, que es un conjunto de 70.000 imágenes pequeñas de dígitos escritas a mano por estudiantes de secundaria y empleados de la Oficina del Centro de Estados Unidos. Cada imagen está etiquetada con el dígito que representa. Este conjunto se ha estudiado tanto que a menudo se le llama el "Hola mundo" del aprendizaje automático: siempre que las personas crean un nuevo algoritmo de clasificación, sienten curiosidad por ver cómo funcionará en MNIST. Siempre que alguien aprende Machine Learning, tarde o temprano se enfrenta al MNIST.

Scikit-Learn proporciona muchas funciones de ayuda para descargar conjuntos de datos populares. MNIST es uno de ellos. El siguiente código recupera el conjunto de datos MNIST: [1]

```
>>> de sklearn.datasets importar fetch_mldata
>>> mnist = fetch_mldata ( 'MNIST original' )
>>> mnist
{'COL_NAMES': ['etiqueta', 'datos'],
  'DESCR': 'mldata.org conjunto de datos: mnist-original', 'datos':
  matriz ([[0, 0, 0, ..., 0, 0, 0],
           [0, 0, 0, ..., 0, 0, 0],
```

1 De forma predeterminada, Scikit-Learn almacena en caché los conjuntos de datos descargados en un directorio llamado $ *INICIO / scikit_learn_data*.