یک اسکریپت برای آموزش مدل بنویسید و به طور خودکار ابرپارامتر‌ها را تنظیم کنید. این اسکریپت می‌تواند به طور خودکار اجرا شود، برای مثال هر روز یا هر هفته، بسته به نیاز شما.

اسکریپت دیگری بنویسید که هم مدل جدید و هم مدل قبلی را در مجموعه آزمایشی به‌روز شده ارزیابی کند و اگر عملکرد آن کمتر نشده بود، مدل را برای تولید پیاده‌سازی کنید (اگر عملکرد آن کمتر شده بود، مطمئن شوید که علت آن را بررسی کرده‌اید). اسکریپت احتمالاً باید عملکرد مدل شما را روی زیرمجموعه‌های مختلف مجموعه آزمایشی، مانند مناطق فقیر یا غنی، مناطق روستایی یا شهری و غیره آزمایش کند.

همچنین باید مطمئن شوید که کیفیت داده‌های ورودی مدل را ارزیابی می‌کنید. گاهی اوقات به دلیل یک سیگنال ضعیف عملکرد کمی کاهش می‌یابد (به عنوان مثال، یک سنسور با عملکرد نادرست که مقادیر تصادفی را ارسال می‌کند، یا خروجی تیم دیگری که به‌روز نباشد)، اما ممکن است مدتی طول بکشد تا عملکرد سیستم شما به اندازه‌ای کاهش یابد که یک هشدار ایجاد شود. اگر ورودی‌های مدل خود را زیر نظر داشته باشید، ممکن است زودتر متوجه این موضوع شوید. به عنوان مثال، اگر ورودی‌های بیشتر و بیشتر یک ویژگی را از دست می‌دهند، یا میانگین یا انحراف استاندارد بیش از حد از مجموعه آموزشی فاصله می‌گیرد، یا یک ویژگی طبقه‌بندی شروع به دسته‌های جدید کند، می‌توانید هشداری را فعال کنید.

در نهایت، در صورتی که مدل جدید به دلایلی شروع به شکست بدی کند، مطمئن شوید که از هر مدلی که ایجاد می‌کنید نسخه‌های پشتیبان تهیه می‌کنید و فرآیند و ابزارهایی را برای بازگشت سریع به مدل قبلی در اختیار دارید. داشتن نسخه پشتیبان نیز امکان مقایسه مدل‌های جدید را با مدل‌های قبلی به راحتی فراهم می‌کند. به طور مشابه، باید از هر نسخه از مجموعه داده‌های خود نسخه پشتیبان تهیه کنید تا در صورت خراب شدن مجموعه داده جدید (به عنوان مثال، اگر معلوم شد که داده‌های جدیدی که به آن اضافه می‌شود پر از موارد پرت است) بتوانید به مجموعه داده قبلی برگردید. داشتن پشتیبان از مجموعه داده‌های خود همچنین به شما امکان می‌دهد هر مدلی را در برابر هر مجموعه داده قبلی ارزیابی کنید.

همانطور که می‌بینید، یادگیری ماشینی شامل زیرساخت‌های بسیار زیادی است. فصل 19 برخی از جنبه‌های این موضوع را مورد بحث قرار می‌دهد، اما موضوع بسیار گسترده‌ای به نام عملیات یادگیری ماشین[[1]](#footnote-2) (MLOps) است که خود می‌تواند یک کتاب باشد. بنابراین تعجب نکنید اگر تولید و بکارگیری اولین پروژه ML شما تلاش و زمان زیادی را در بر داشته باشد. خوشبختانه، زمانی که تمام زیرساخت‌‌ها فراهم شود، انتقال از ایده به تولید بسیار سریعتر خواهد بود.

امتحان کنید!

امیدواریم این فصل به شما ایده خوبی از ظاهر یک پروژه یادگیری ماشینی داده و همچنین برخی از ابزارهایی را که می‌توانید برای آموزش یک سیستم عالی استفاده کنید، به شما نشان دهد. همانطور که می‌بینید، بیشتر کار در مرحله آماده‌سازی داده‌‌ها است: ساخت ابزار نظارت، راه اندازی فرآیند ارزیابی انسانی، و خودکارسازی آموزش مدل‌های منظم. هرچند الگوریتم‌های یادگیری ماشین مهم هستند، اما احتمالاً ترجیح داده می‌شود که با فرآیند کلی مورد بررسی قرار گرفته و سه یا چهار الگوریتم را به خوبی بشناسید تا اینکه تمام وقت خود را صرف بررسی الگوریتم‌های پیشرفته کنید.

بنابراین، اگر قبلاً این کار را انجام نداده‌اید، اکنون زمان خوبی است که یک کامپیوتر را انتخاب کرده، مجموعه داده‌ای را انتخاب کنید که به آن علاقه‌مند هستید و سعی کنید کل فرآیند را از ابتدا تا انت‌ها طی کنید. نقطه مناسبی برای شروع می‌تواند یک وب سایت مسابقه مانند Kaggle باشد که شما یک مجموعه داده برای بازی، یک هدف واضح و افرادی برای به اشتراک گذاشتن تجربه خواهید داشت. از آن لذت ببرید!

تمرینات

تمرین‌های زیر بر اساس مجموعه داده مسکن موجود در این فصل است:

1. یک رگرسیور ماشین بردار پشتیبان[[2]](#footnote-3) (sklearn.svm.SVR) با هایپرپارامترهای مختلف مانند kernel="linear" (با مقادیر مختلف برای هایپرپارامتر C) یا kernel="rbf" (با مقادیر مختلف برای هایپرپارامترهای C و گاما) امتحان کنید. توجه داشته باشید که ماشین‌های بردار پشتیبان به خوبی به مجموعه داده‌های بزرگ، مقیاس نمی‌شوند، بنابراین احتمالاً باید مدل خود را فقط در 5000 نمونه اولیه مجموعه آموزشی، آموزش دهید و فقط از اعتبارسنجی متقاطع 3 برابری استفاده کنید، در غیر این صورت ساعت‌‌ها طول خواهد کشید. در حال حاضر نگران معنای هایپرپارامتر‌ها نباشید. ما آن‌‌ها را در فصل 5 مورد بحث قرار خواهیم داد. بهترین‌پیش بینی کننده SVR چگونه عمل می‌کند؟

2. سعی کنید GridSearchCV را با RandomizedSearchCV جایگزین کنید.

3. سعی کنید یک تبدیل‌کننده SelectFromModel را در فرآیند آماده‌سازی اضافه کنید تا فقط مهم‌ترین ویژگی‌‌ها را انتخاب کنید.

4. سعی کنید یک تبدیل‌کننده سفارشی ایجاد کنید که رگرسیور k نزدیکترین همسایه (sklearn.neighbors.KNeighborsRegressor) را در روش fit() خود آموزش دهد و پیش‌بینی‌های مدل را در روش transform() خود خروجی دهد. سپس این ویژگی را با استفاده از طول و عرض جغرافیایی به عنوان ورودی این تبدیل‌کننده به فرآیند پیش‌پردازش اضافه کنید. این یک ویژگی را در مدل اضافه می‌کند که با میانگین قیمت مسکن نزدیک‌ترین مناطق مطابقت دارد.

5. به طور خودکار برخی از گزینه‌های آماده سازی را با استفاده از GridSearchCV کاوش کنید.

6. سعی کنید دوباره کلاس StandardScalerClone را از ابتدا پیاده‌سازی کنید، سپس پشتیبانی به روش inverse\_transform() را اضافه کنید: کلاس scaler را اجرا کنید.

inverse\_transform(scaler.fit\_transform(X)) باید آرایه‌ای بسیار نزدیک به X برگرداند. سپس پشتیبانی از نام ویژگی‌‌ها را اضافه کنید: اگر ورودی DataFrame باشد feature\_names\_in\_ را در روش fit() تنظیم کنید. این ویژگی باید یک آرایه NumPy از نام ستون‌‌ها باشد. در آخر، روش get\_feature\_names\_out() را پیاده‌سازی کنید: باید یک آرگومان اختیاری input\_features=None argument داشته باشد. در صورت تصویب، روش باید بررسی کند که طول آن با n\_features\_in\_ مطابقت داشته باشد و اگر تعریف شده باشد باید با feature\_names\_in\_ مطابقت داشته باشد. سپس input\_features باید برگردانده شوند. اگر input\_features هیچ باشد، در این صورت روش باید feature\_names\_in\_ را در صورت تعریف برگرداند یا در غیر این صورت np.array(["x0", "x1", ...])، با طول n\_features\_in\_.

راه‌حل‌های تمرین‌های انتهای این فصل در https://homl.info/colab3 موجود است.

1 مجموعه داده اصلی در R. Kelley Pace و Ronald Barry موجود است شد:

“Sparse Spatial Autoregressions”, *Statistics & Probability Letters* 33, no. 3 (1997): 291–297.”

2 بخشی از اطلاعاتی که به سیستم یادگیری ماشینی داده می‌شود، اغلب سیگنال نامیده می‌شود، با اشاره به نظریه اطلاعات کلود شانون، که او در آزمایشگاه‌های بل برای بهبود ارتباطات راه دور ایجاد کرد. نظریه او: شما یک نسبت سیگنال به نویز بالا می‌خواهید.

3 به یاد بیاورید که عملگر transpose یک بردار ستون را به یک بردار ردیف تبدیل می‌کند (و بالعکس).

4 همچنین ممکن است لازم باشد محدودیت‌های قانونی را بررسی کنید، مانند فیلدهای خصوصی که هرگز نباید در فروشگاه‌های داده ناامن کپی شوند.

5 انحراف استاندارد به طور کلی با σ (حرف یونانی سیگما) نشان داده می‌شود، و جذر واریانس است که میانگین مجذور انحراف از میانگین است. هنگامی که یک ویژگی دارای توزیع نرمال زنگ ‌شکل است (که توزیع گاوسی نیز نامیده می‌شود و بسیار رایج است)، قانون «99.7-95-68» اعمال می‌شود: حدود 68 درصد از مقادیر در σ1، 95 درصد در σ2 و 99.7% در σ3 از میانگین قرار می‌گیرند.

6 اغلب می‌بینید که افراد هسته تصادفی را روی 42 تنظیم می‌کنند. این عدد هیچ ویژگی خاصی ندارد، به جز پاسخ به سؤال نهایی زندگی، جهان و همه چیز.

7 اطلاعات مکان در واقع کاملاً بزرگ است، و در نتیجه بسیاری از مناطق دقیقاً همان شناسه را خواهند داشت، بنابراین در همان مجموعه (آزمون یا آموزش) قرار می‌گیرند. این مورد سوگیری نامناسبی را نمونه‌گیری وارد می‌کند.

8 اگر این را در مقیاس خاکستری می‌خوانید، یک خودکار قرمز بردارید و روی بیشتر خط ساحلی از منطقه خلیج تا سن دیگو (همانطور که انتظار دارید) خط خطی کنید. می‌توانید یک تکه رنگ زرد در اطراف ساکرامنتو نیز اضافه کنید.

9 برای جزئیات بیشتر در مورد اصول طراحی به مرجع زیر مراجعه کنید:

“API Design for Machine Learning Software: Experiences from the Scikit-Learn Project”, arXiv preprint arXiv:1309.0238 (2013).

10 برخی از پیش بینی ‌ها نیز روش هایی را برای اندازه گیری اطمینان پیش بینی‌های خود ارائه می‌دهند.

11 تا زمانی که این سطر‌ها را می‌خوانید، ممکن است همه تبدیل‌کننده‌‌ها زمانی که یک DataFrame را به عنوان ورودی دریافت می‌کنند، خروجی DataFrames Pandas را ایجاد کنید: Pandas in, Pandas out. احتمالاً یک گزینه پیکربندی جهانی برای این وجود خواهد داشت: sklearn.set\_config(pandas\_in\_out=True).

12 برای جزئیات بیشتر به اسناد SciPy مراجعه کنید.

13 به طور خلاصه، یک API REST (یا RESTful) یک API مبتنی بر HTTP است که از برخی قرارداد‌ها مانند استفاده از افعال استاندارد HTTP برای خواندن، به‌روزرسانی، ایجاد یا حذف منابع (GET، POST، PUT، و DELETE) و استفاده از JSON برای ورودی و خروجی پیروی می‌کند.

14 کپچا تستی است برای اطمینان از اینکه کاربر ربات نیست. این تست‌‌ها اغلب به عنوان روشی ارزان برای برچسب گذاری داده‌های آموزشی مورد استفاده قرار گرفته‌اند.

**فصل 3. طبقه‌بندی**

در فصل 1 اشاره کردیم که رایج‌ترین وظایف یادگیری تحت نظارت، رگرسیون (پیش‌بینی مقادیر) و طبقه‌بندی (پیش‌بینی کلاس‌ها) است. در فصل 2 ما یک کار رگرسیون را بررسی کردیم، پیش‌بینی مقادیر مسکن، با استفاده از الگوریتم‌های مختلف مانند رگرسیون خطی، درخت‌های تصمیم‌گیری، و جنگل‌های تصادفی (که در فصل‌های بعدی با جزئیات بیشتر توضیح داده خواهد شد). اکنون توجه خود را به سیستم‌های طبقه‌بندی معطوف خواهیم کرد.

**MNIST**

در این فصل ما از مجموعه داده MNIST استفاده خواهیم کرد که مجموعه ای از 70000 تصویر کوچک از ارقام است که توسط دانش‌آموزان دبیرستانی و کارمندان اداره سرشماری ایالات متحده نوشته شده است. هر تصویر با رقمی که نشان می‌دهد برچسب‌گذاری می‌شود. این مجموعه به قدری مورد مطالعه قرار گرفته است که اغلب به آن «سلام جهان» یادگیری ماشینی می‌گویند: هر زمان که افراد الگوریتم طبقه‌بندی جدیدی را ارائه می‌کنند، کنجکاو هستند که ببینند چگونه در MNIST کار می‌کند، و هرکسی که یادگیری ماشینی را یاد می‌گیرد با این دیتاست دیر یا زود روبه‌رو می‎‌شود.

Scikit-Learn بسیاری از توابع کمکی را برای دانلود مجموعه داده‌های محبوب فراهم می‌کند که MNIST یکی از آن‌هاست. کد ارائه شده در ذیل، مجموعه داده MNIST را از OpenML.org واکشی می‌کند:

**from sklearn.datasets import** fetch\_openml

mnist = fetch\_openml('mnist\_784', as\_frame=**False**)

بسته sklearn.datasets عمدتا شامل سه نوع تابع است: توابع fetch\_\* مانند fetch\_openml() برای دانلود مجموعه داده‌های واقعی، توابع load\_\* برای بارگیری مجموعه داده‌های اسباب بازی کوچک همراه با Scikit-Learn (بنابراین نیازی به دانلود از طریق اینترنت ندارند.)، و توابع make\_\* برای تولید مجموعه داده‌های جعلی، مفید برای آزمایش. مجموعه داده‌های تولید شده معمولاً به صورت یک تاپل (X, y) حاوی داده‌های ورودی و اهداف، هر دو به عنوان آرایه NumPy، بازگردانده می‌شوند. سایر مجموعه‌های داده به‌عنوان اشیاء sklearn.utils.Bunch بازگردانده می‌شوند، که فرهنگ‌های لغت هستند که ورودی‌های آن‌‌ها نیز می‌تواند به عنوان ویژگی قابل دسترسی باشد. آن‌ها به طور کلی شامل ورودی‌های زیر هستند:

توصیفی از مجموعه داده:"DESCR"

داده‌های ورودی، معمولاً به صورت یک آرایه 2 بعدی NumPy: *"data"*

برچسب ‌ها معمولاً به صورت آرایه 1 بعدی NumPy هستند: *"target"*

تابع fetch\_openml() کمی غیرعادی است زیرا به طور پیش‌فرض ورودی‌‌ها را به صورت Pandas DataFrame و برچسب‌‌ها را به صورت Pandas Series برمی‌گرداند (مگر اینکه مجموعه داده پراکنده باشد). اما مجموعه داده MNIST حاوی تصاویر است و DataFrames برای آن ایده‌آل نیستند، بنابراین ترجیح داده می‌شود که as\_frame=False را تنظیم کنید تا داده‌‌ها را به‌عنوان آرایه‌های NumPy دریافت کنید. بیایید به این آرایه‌‌ها نگاه کنیم:

**>>>** X, y = mnist.data, mnist.target **>>>** X

array([[0., 0., 0., ..., 0., 0., 0.],

[0., 0., 0., ..., 0., 0., 0.],

[0., 0., 0., ..., 0., 0., 0.],

...,

[0., 0., 0., ..., 0., 0., 0.],

[0., 0., 0., ..., 0., 0., 0.],

[0., 0., 0., ..., 0., 0., 0.]])

**>>>** X.shape

(70000, 784)

**>>>** y

array(['5', '0', '4', ..., '4', '5', '6'], dtype=object)

**>>>** y.shape

(70000,)

70000 تصویر وجود دارد و هر تصویر دارای 784 ویژگی است. این به این دلیل است که هر تصویر 28 × 28 پیکسل است و هر ویژگی به سادگی نشان دهنده شدت یک پیکسل است، از 0 (سفید) تا 255 (سیاه). بیایید نگاهی به یک رقم از مجموعه داده بیندازیم (شکل 3-1). تن‌ها کاری که باید انجام دهیم این است که بردار ویژگی یک نمونه را بگیریم، آن را به یک آرایه ۲۸×۲۸ تغییر شکل دهیم و با استفاده ازMatplotlib imshow () function آن را نمایش دهیم. ما از cmap="binary" برای بدست آوردن یک نقشه رنگی خاکستری استفاده می‌کنیم که در آن 0 سفید و 255 سیاه است:

**import matplotlib.pyplot as plt**

**def** plot\_digit(image\_data):

image = image\_data.reshape(28, 28)

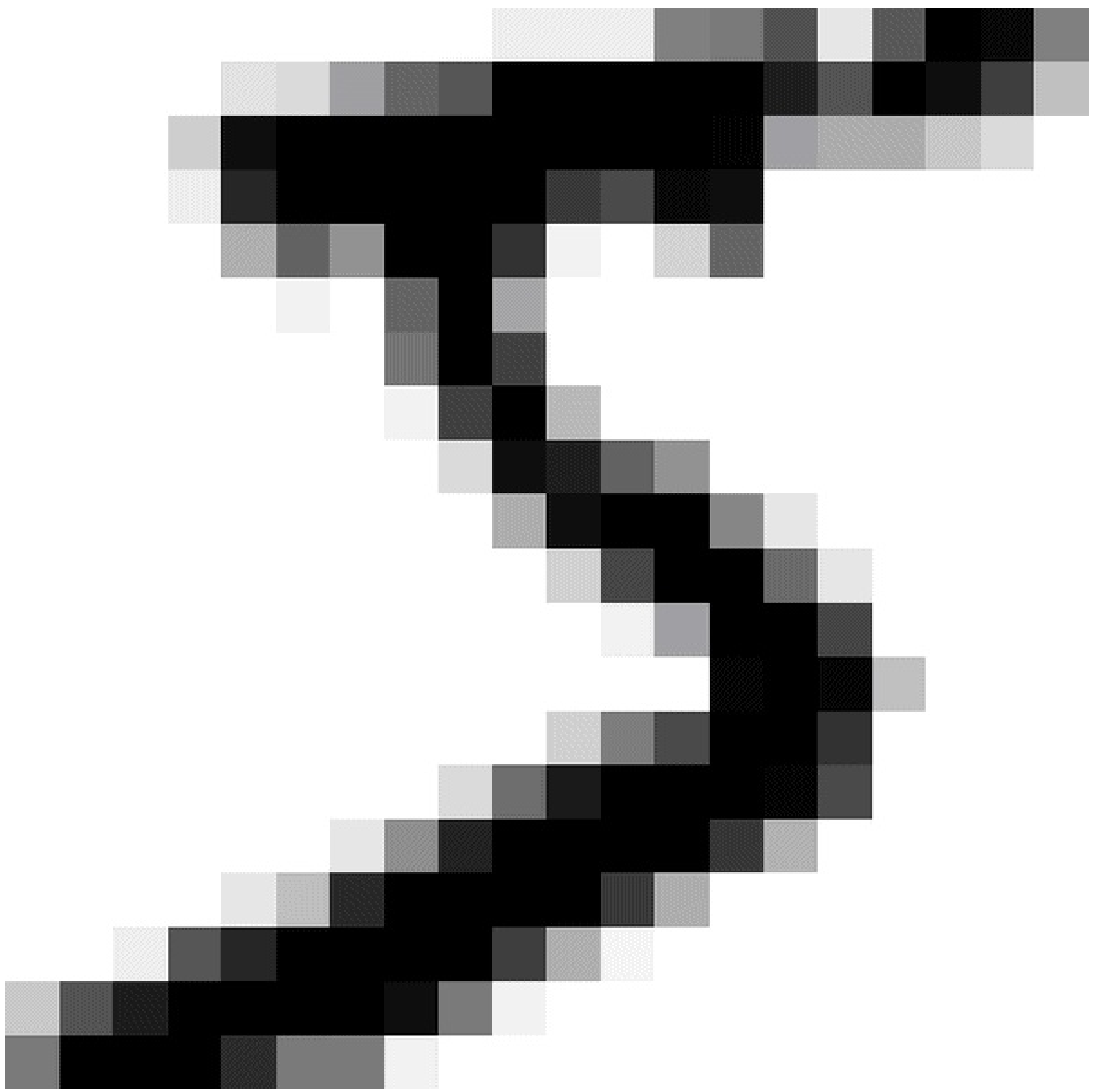
plt.imshow(image, cmap="binary")

plt.axis("off")

some\_digit = X[0]

plot\_digit(some\_digit)

plt.show()



شکل 3-1. نمونه ای از تصویر MNIST

این شبیه 5 است، و در واقع این چیزی است که برچسب به ما می‌گوید:

**>>>** y[0]

'5'

1. Machine Learning Operations [↑](#footnote-ref-2)
2. Support Vector Machine Regressor [↑](#footnote-ref-3)