به نام خدا

خلاصه فصل اول کتاب

HANDS-ON MACHINE LEARNING

ماشین لرنینگ چیه؟

- به فیلدی که به کامپیوتر ها قابلیت این رو میده که بدون اینکه بطور صریح برنامه نویسی شده باشن، توانایی یادگیری رو داشته باشن، ماشین لرنینگ میگن.
- برای این کار ابتدا نیاز داریم یه سری دیتا داشته باشیم تا ازشون یادبگیریم . به مجموعه کلی دیتاهامون training set و به هر عضو ان یک sample گفته میشه. و به قسمتی از از سیستم که توانایی یادگیری و پیش‌بینی داره model میگن. برای مثال اگه بخوایم یه سیستم که توانایی تشخیص بده که یه ایمیل رو filter کنیم یا نه به طور کلی به کلی ایمیل نیاز داریم که داخلش یه سری ایمیل filter و یه سری ایمیل ham داشته باشه (لیبل گذاشته باشیم رو ایمیلامون که کامپیوتر بفهمه هر ایمیل از چه نوعیه).
- چرا باید از ماشین لرنینگ استفاده کنیم؟ بیایم همین سیستم که تشخیص نوع ایمیل میکنه رو ببریم رو سیستم های قدیمی. تو این سیتم ها باید خودتون تشخیص بدین که عموم این ایمیل ها شامل چه کلماتی هستن. مثلا شامل یه سری کلمه مثل credit card, 4U, free .. باشه. بعد هر ایمیل که میاد میبینه چه مقدار از این کلمات رو داره و تشخیص بده که ایمیل به چه دسته ای تعلق داره.
- اولین بدی این روش اینه که برای همچین کاری باید کلی قانون بنویسیم و در ادامه اگه به جای 4U بنویسن For U قراره اشتباه کنه.

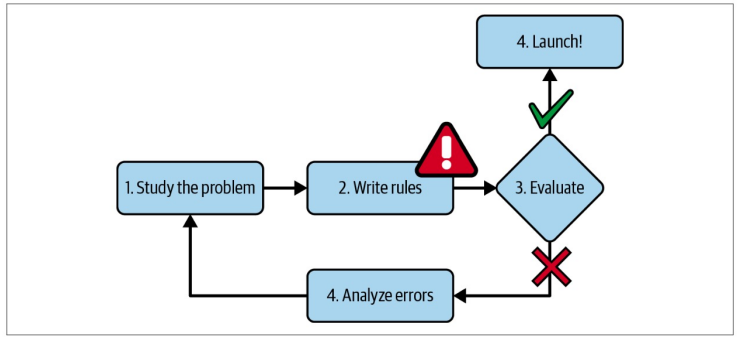

Figure 1-1. The traditional approach

ادامه

- در عوض یک سیستم که بر اساس ماشین لرنینگ توسعه یافته به جای این کار خودش تشخیص میده چه کلماتی اهمیت بیشتری داره و بر اساس تغییرات جدید خودش رو میتونه آپدیت کنه.

یه جای دیگه که میایم از ماشین لرنینگ استفاده میکنیم جایی که برای یک مسئله که پیچیدگی زیادی برای پیاده سازی دارن یا اینکه الگوریتم مشخصی نداره. تو این قسمت اجازه میدیم خود ماشین شروع کنه به یادگیری و پیداکردن راه حل.

یه مشخصه خوب دیگه سیستم های ماشین لرنینگ اینه که بعد از فرایند یادگیری بهمون بگن چی یادگرفتن. برای مثال تو همون سیستم تشخیص نوع ایمیل بعد از فرایند یادگیری میتونیم ببینیم مدلمون به چه کلماتی حساس شده برای تشخیص نوع ایمیلمون. یا اینکه یه سری کلمات رو تشخیص بده که با هم ارتباط داشتن. به این فرایند که از یک سری دیتای خیلی بزرگ میایم یه سری پترن های مخفی رو پیدا میکنیم data mining میگن.

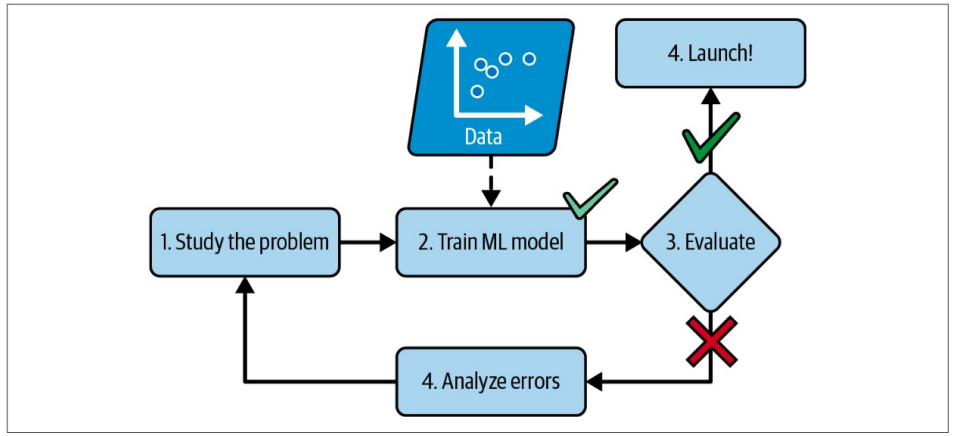

Figure 1-2. The machine learning approach

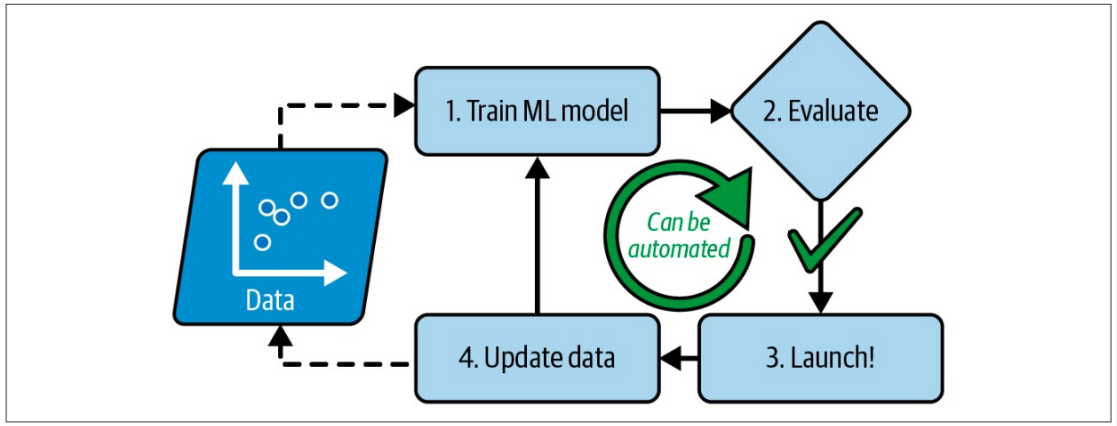

Figure 1-3. Automatically adapting to change

انواع سیستم های ماشین لرنینگ

- به طور کلی میتونیم همچین دسته بندی داشته باشیم:
- ۱. برحسب نظارتی که روی traning set داریم میشه به: supervised, unsupervised, semi-supervised, self-supervised و ... تقسیم بندی کرد.
- ۲. online learning vs batch learning (جلوتر توضیح میدیم)
- ۳. بر اساس اینکه یک دیتا جدید برای پیش بینی وقتی میاد، با مقایسه با دیتاهای training set میایم جواب رو پیش‌بینی میکنیم یا اینکه بیایم بر اساس با تشخیص پترن هایی که مدل یادگرفته این کارو انجام بدیم.
- Supervised learning: تو این روش دیتاست باید شامل یک سری دیتا لیبل دار باشه. یعنی باید مشخص کنیم هر نمونه چه خروجی داره. برای مثال تو سیستم تشخیص ایمیل هر سمپل شامل لیبل اینکه spam یا ham هستش رو داشته باشه. یک کاربرد کلی همچین سیستمی classification هستش. داخل این سیستم یک نمونه جدید رو برای پیش بینی به یه کلاس classify میکنیم. یک کاربرد دیگه اینه به جای اختصاص به یک کلاس بیایم یک مقدار رو پیش‌بینی کنیم. مثل زمانی که مدلمون کارش پیشبینی قیمت یه خونه بر اساس یک سری ویژگی مثل متراژ، تعداد اتاق خواب و ... هست که در اون صورت تسک regression هست نه classify. گرچه میشه از مدل های regression مثل logistic regression برای classify کردن هم استفاده کرد یا برعکس (این مدل میاد یه عدد بین ۰ و ۱ که میزان احتمال تعلق به یه کلاسه به عنوان عدد تولید میکنه)

ادامه

- Unsuperviesd: تو این مدل بر خلاف مدل قلبی training set سمپل های داخلش دارای لیبل نیست. یه مثال برای کاربردش میتونه این شکلی باشه که کلی بازدید کننده از یه سایت داری و میخوای بدونی معمولا چه نوع ادم هایی هستن که میان تو این سایت و میخوایم اونارو به گروه های نزدیک به هم clutser بندی کنیم.

At no point do you tell the algorithm which group a visitor belongs to: it finds those connections without your help. For example, it might notice that 40% of your visitors are teenagers who love comic books and generally read your blog after school, while 20% are adults who enjoy sci-fi and who visit during the weekends. If you

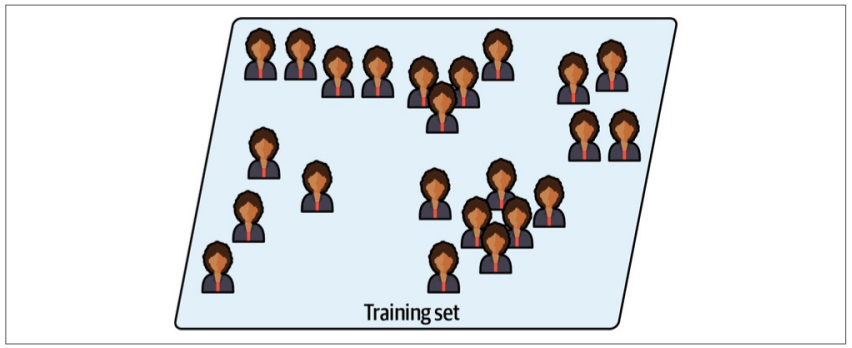

Figure 1-7. An unlabeled training set for unsupervised learning

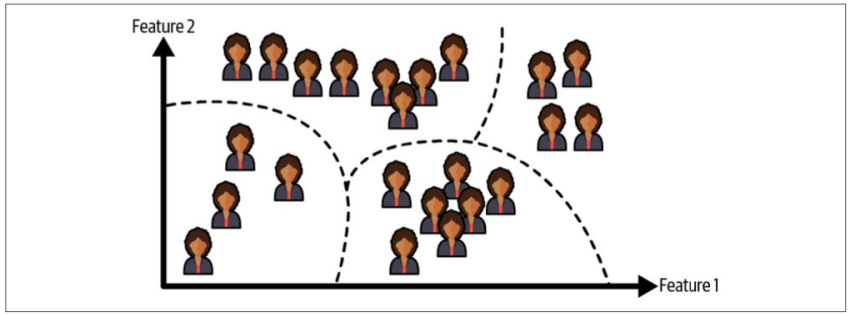

Figure 1-8. Clustering

یک تسک مرتبط دیگه dimension reduction هستش که به این شکل هست که دیتا ها رو به شکلی که اطلاعت خیلی مهمی ازشون از بین نره یه سری اطلاعاتشون رو حذف کنیم. یک روش برای این کار feature extraction هست که به این شکله میاد میبینه که یک سری فیچر شبیه به همن، مثل سن خودرو و میزان مسافتی که طی کرده و این دو تا فیچر رو با یه فیچر که بتونه یک معیار از این دوتا باشه جایگزین میکنه. از خوبیای این کار اینه اول از همه فضا کمتری برای نگه داری دیتا نیاز داریم دوم هم اینکه فرایند یادگیری سریع تر میشه چون مدل با فیچر های کمتری سروکار داره.

ادامه

- تسک بعدی که میشه انجام داد با unsupervised learning، anomaly detection هست. که برای مثال یه کاربرد اون اینه بیایم یه تراکنش غیرعادی رو تشخیص بدیم یا حتی تو فرایند یادگیری بیایم training set رو از یه سری دیتا پرت (outlier) پاکسازی کنیم. وقتی یه نمونه جدید وارد بشه میتونه تشخیص بده با دیتاهایی که داخل دیتاست هست شباهت داره یا نه و از روی اون تشخیص بده anomal هست یا نه.

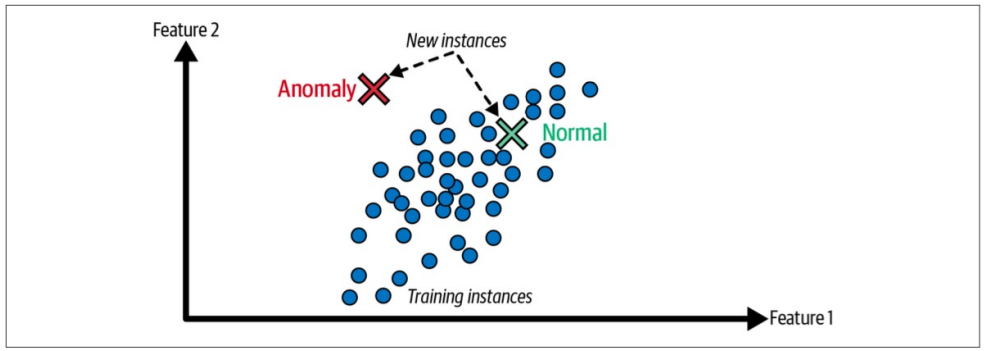

Figure 1-10. Anomaly detection

- یک جای دیگه که unsupervised بهمون کمک میکنه association rule learning هست که هدفش اینه داخل یه سری دیتا خیلی بزرگ بیاد یه سری روابط خاص رو پیدا کنه و بهمون بگه. برای مثال بگه اونایی که معمولا سس و سیب‌زمینی از یه فروشگاه خریدن معمولا استیک هم خریدن.

ادامه

- Semi-supervised learning: از اونجایی که لیبل زدن به دیتاها معمولا کار طولانی هستش به جای اینکه همه دیتاها رو لیبل گذاری کنیم، میایم یه سری از دیتاها رو صرفا لیبل‌گذاری میکنیم.

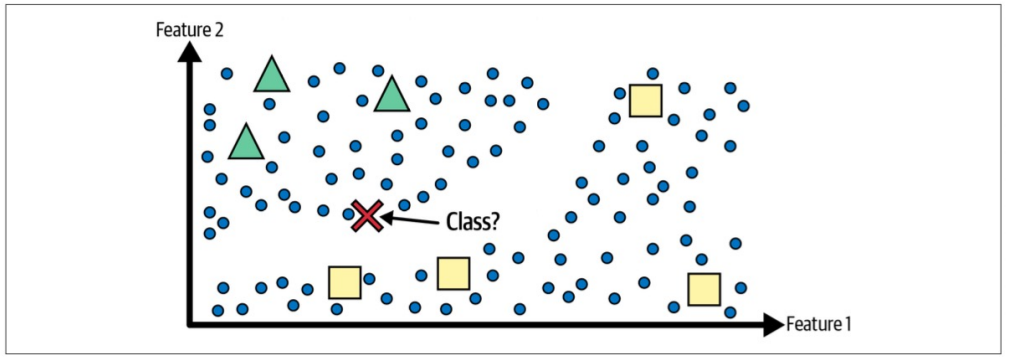

Figure 1-11. Semi-supervised learning with two classes (triangles and squares): the unlabeled examples (circles) help classify a new instance (the cross) into the triangle class rather than the square class, even though it is closer to the labeled squares

- اکثر شکل این سیستم به این شکله اول از یه unsupervised استفاده میکنیم و دیتاهای شبیه رو کلاستر بندی کنیم و بعدش از بخشsupervised کمک بگیریم و هر کلاستر رو خودمون لیبل گذاری کنیم. (کاری که google photos میکنه همینه که بیاد تشخیص چهره کنه داخل عکس های مختلف و از شما بپرسه اخرش اینا چه افرادین؟)

ادامه

- self-supervised learning: کاری که تو این بخش انجام میدیم اینه بهمون یه دیتاست که هیچ لیبلی نداره بهمون میدن و در یه بخش از دیتا رو هاید میکنیم تا مدل اول بیاد اون بخشی که هاید شده رو درست کنه (اینطوری یادمیگیره هر دیتا چه ویژگی داره) بعد که دیتاها رو ساخت حالا لیبل دارن و میتونیم از متد های supervised استفاده کنیم.

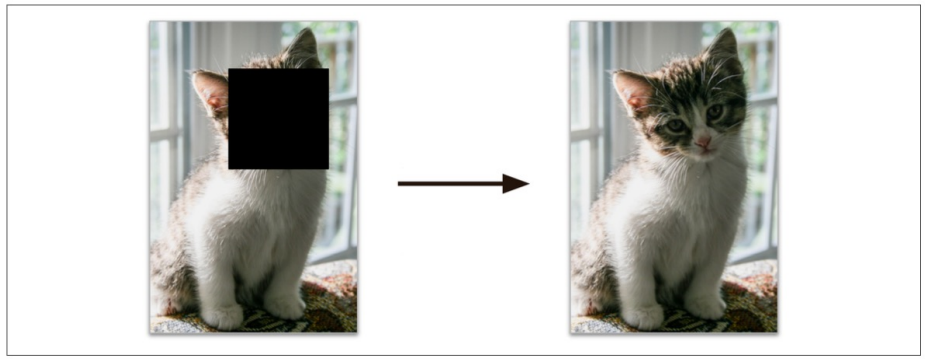

Figure 1-12. Self-supervised learning example: input (left) and target (right)

فرض کنید شما یک مدل که حیون های مختلف رو کلاس بندی میکنه میخواید داشته باشید. برای این کار یه دیتاست بزرگ که کلی عکس حیون داخلش هست رو داریم. اول کاری که میکنیم میایم مدل رو برای کامل کردن عکس حیون ها train میکنیم بعدش که مدل جاهای مختلف حیون رو یادبگیره میتونه بفهمه هر حیون چه ویژگی هایی داره و کلاس بندیش میکنه.

- حالا با یه ذره تغییر مدل روی دیتاهای لیبل دار به هدف اصلیمون میرسیم. به این کار که بیایم یه مدل رو اول برای یه کار دیگه train کنیم بعدش از تجربیاتش به یه کار دیگه استفاده کنیم transfer learning میگیم. که به طور زیادی داخل شبکه های عصبی انجام میشه.

ادامه

- Reinforcment learning: داخل این فرایند به سیستم یادگیری agent میگیم. میتونه با محیط اطرافش در تعامل باشه تصمیم بگیره و فیدبک بگیره ازمون که شامل تشویق و تنبیه میشه. در نهایت با توجه به سیاست هایی که یادمیگیره هدفش اینه بیشترین تشویق رو در طول فراید بدست بیاره.

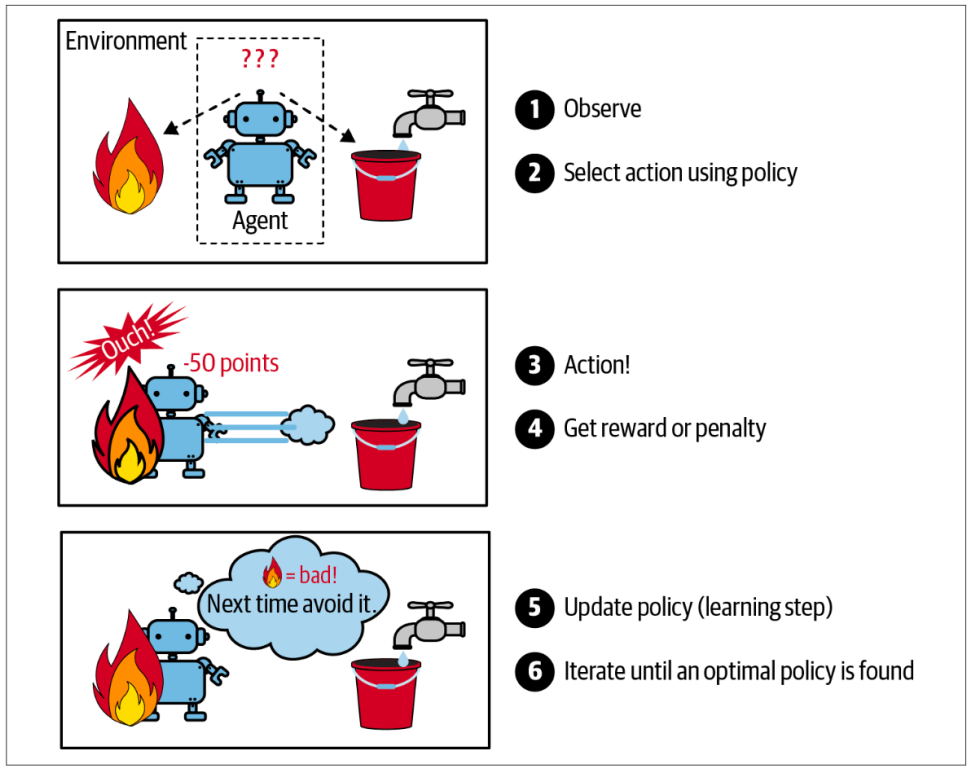

Figure 1-13. Reinforcement learning

For example, many robots implement reinforcement learning algorithms to learn how to walk. DeepMind's AlphaGo program is also a good example of reinforcement learning: it made the headlines in May 2017 when it beat Ke Jie, the number one ranked player in the world at the time, at the game of Go. It learned its winning policy by analyzing millions of games, and then playing many games against itself. Note that learning was turned off during the games against the champion; AlphaGo was just applying the policy it had learned. As you will see in the next section, this is called *offline learning*.

BATCH LEARNING VS ONLINE LEARNING

- یک معیار دیگه برای تقسیم بندی سیستم ها به این دو دست هستش:
- Batch learning: به این روش offline learning هم گفته میشه و به این شکله که باید یک سری دیتا کلی بهش بدیم اول، یادبگیره ازشون و بعد که آماده شد طبق همون دیتاهای قبلی بدون اضافه شدن و یادگیری جدید بره برای پیش‌بینی و از نمونه های جدید که برای پیش‌بینی میاد هم چیزی یادنمیگیره. معمولا از لحاظ زمانی روش زمان بریه چون باید اولش کلی فرایند یادگیری انجام بده. بدی همچین مدل هایی اینه که پرفورمنس به تدریج کم میشه چون دیتاها با گذر زمان داخل دنیا درحال تغییرن. راه حل برای این مشکل اینه که هرچند وقت یه بار دوباره مدلمون رو دیتاهای جدید train کنیم.
  مدل باید روی دیتاها قدیمی و جدید train بشه.

Even a model trained to classify pictures of cats and dogs may need to be retrained regularly, not because cats and dogs will mutate overnight, but because cameras keep changing, along with image formats, sharpness, brightness, and size ratios. Moreover, people may love different breeds next year, or they may decide to dress their pets with tiny hats—who knows?

یه بدی دیگه ای که این روش داره اینه اگه یه سیستم predict stock price داشته باشیم باید هر روز با دیتاهای جدید train کنم و حجم اطلاعات انقدر زیاده ممکنه فرایند یادگیری کلی computing resource مثل cpu نیاز داشته باشه و کلی وقت هم ببره پس به یه روشی نیاز داریم که بتونه این مشکلات رو حل کنه.

ONLINE LEARNING

۲. online learning: تو این روش هر sampleی که بهش پاس میدیم با همون یه نمونه میاد یادگیری رو انجام میده و خودشو آپدیت میکنه. یا اینکه جای اینکه متکی به یه sample باشه، وابسته به تعداد کمی از sampleهاست که بهش میگن mini-batch معمولا سایزشون ۳۲تاییه که یعنی داخل هر mini-batch، ۳۲ تا sample هستش و بعد از تموم شدن هر mini-batch مدلمون خودشو آپدیت میکنه.

این روش خوبیش اینه به ازای موقعیت هایی که باید به طور سریع خودشو اپدیت کنه با دیتاهای جدید، مثلا detect new patterns in the stock market. یا اینکه سیستممون از منابع محاسباتی محدودی بهره میبره که باعث میشه نتونه کلی دیتا رو مثلا داخل ram خودش یه جا نگه داره چون کل traning set رو تقسیم به یه سری mini-batch کردیم و هر سری با اون train کردیم مدل رو.

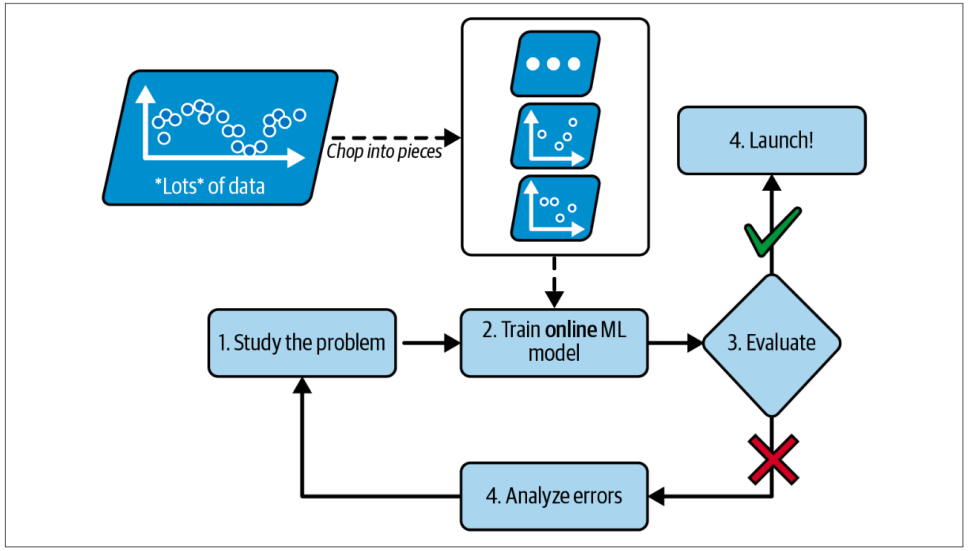

Figure 1-15. Using online learning to handle huge datasets

یه پارامتر مهم داخل این مدل یادگیری، learning rate هست که یک معیاری برای اینکه مدلمون تا چه حدی خودشو با دیتاهای جدید بخواد تغیر بده هستش. اگه این پارامتر خیلی بزرگ باشه باعث میشه مدل سریع به دیتاهای جدید خودشو وفق بده و دیتاهای قبلیو یادش بره و برعکس. یه نکته مهم اینه ممکنه یه دیتای پرت بیاد داخل مدلمون و اگه learning rate بالایی داشته باشیم مدل performanceش میاد پایین.

INSTANCE-BASED VS MODEL-BASED LEARNING

- تسک اصلی مدلمون اینه بهش یه سری دیتا بدیم ازشون یادبگیره و بتونه رو دیتاهای جدید پیش‌بینی انجام بده. اینکه چطوری generalize میکنن خودش یک دسته بندی میتونه باشه برامون.
- ۱.instance-based learning : تو این روش نحوه پیش‌بینی ما خیلی متکی به دیتاهایی که از قبل دیدیم. برای مثال یه سیستم تشخیص filter ایمیل میتونه این شکلی باشه بهش کلی ایمیل spam و ham بدیم و هر نمونه جدیدی که میاد ببینیم به کدوم یکی از دیتاهامون شبیه هستش و طبق اون تصمیم گیری میکنه. در واقع تو اینجا ما به یه معیار شباهت یا measuer of similarity نیاز داریم تا بتونیم شباهت رو بسنجیم. مثلا برای سیستم ایمیل میتونه تعداد کلمات مشابه باشه.

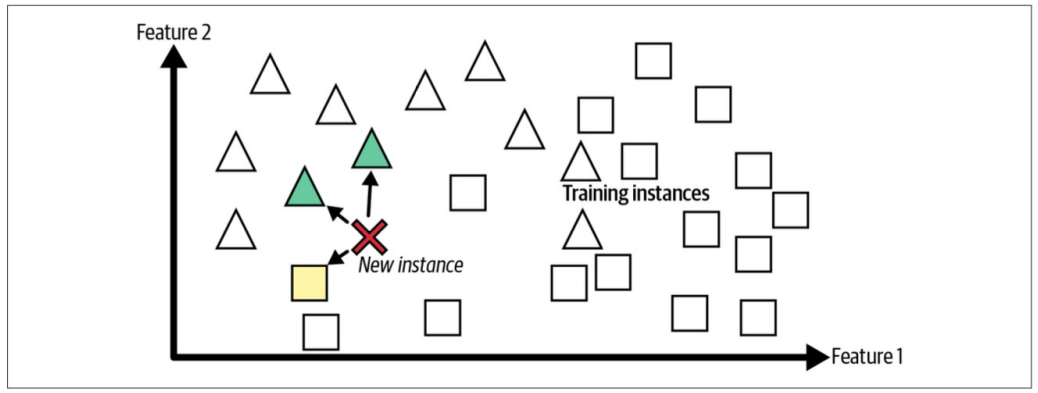

Figure 1-16. Instance-based learning

MODEL-BASED LEARNING

- ۲. model-based learning: فرض کنید اگه بخوایم یه مدلی که میزان خوشحالی یک مردم رو بر اساس معیار GDP بسنجیم. اول دیتامون رو پلات میکنیم ببینم چی ازش متوجه میشیم؟

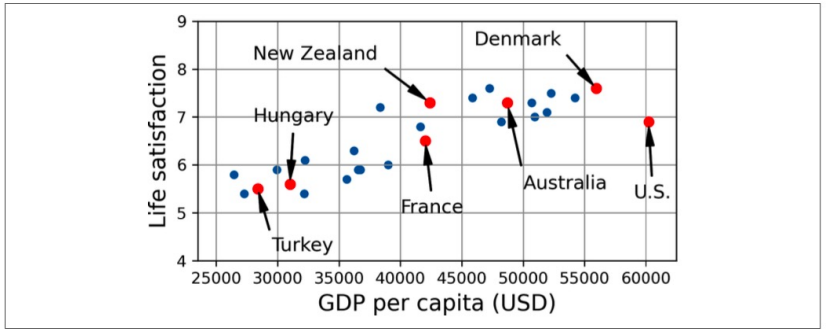

Figure 1-18. Do you see a trend here?

- آفرین! پول خوشبختی میاره. علاوه بر اون یه ترند خطی رو داخلش میبینیم پس یه مدل خطی رو براش انتخاب میکنیم.

Equation 1-1. A simple linear model

$$\text{life\_satisfaction} = \theta_0 + \theta_1 \times \text{GDP\_per\_capita}$$

- این مدل همینطور که از تساوی بالا میبینیم باید دو تا پارامتر رو پیدا کنه. تتاه و تتا۱. با انتخاب مقدار های مختلف میتونیم خط های مختلف رو بسازیم. اما نکته اینه چطوری؟ نیاز به یه معیار داریم که بتونه هر مدل performanceش رو بسنجه. این معیار برای ما یهcost function هستش که بیاد ببینه هر مدل چه میزان خطاهایی داشته و سعی کنیم این تابع رو مینیمم کنیم. با این کار میتونیم دو تا پارامتر تتا۱ و تتاه رو پیدا کنیم.

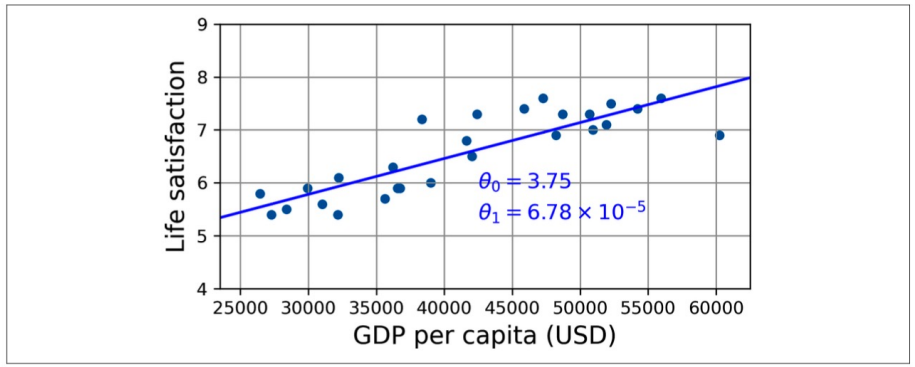

Figure 1-20. The linear model that fits the training data best

ادامه

- حالا که این خط رو داریم میتونیم بهش ورودی بدیم و خروجی بگیریم. خیلی کار سختی هم نیست بخوایم همچین کدی بزنیم:

اگه میخواستیم به جای model-based از instance-based استفاده میکردیم نگاه میکردیم ببینیم عدد ورودیمون به کدوم یکی از عدد های training-set نزدیک تره تا بتونه پیش بینی رو انجام بده ولی بدیش اینه برای این کار شما باید همه training-set رو داخل ram نگه داری کنی تا هر نمونه جدید میاد بتونی سریع دیتا رو ازش بخونی و تابع شباهتت رو فراخونی بکنی و خروجی رو بدی.

```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression

# Download and prepare the data
data_root = "https://github.com/ageron/data/raw/main/"
lifesat = pd.read_csv(data_root + "lifesat/lifesat.csv")
X = lifesat[["GDP per capita (USD)"]].values
y = lifesat[["Life satisfaction"]].values

# Visualize the data
lifesat.plot(kind='scatter', grid=True,
             x="GDP per capita (USD)", y="Life satisfaction")
plt.axis([23_500, 62_500, 4, 9])
plt.show()

# Select a linear model
model = LinearRegression()

# Train the model
model.fit(X, y)

# Make a prediction for Cyprus
X_new = [[37_655.2]]  # Cyprus' GDP per capita in 2020
print(model.predict(X_new)) # output: [[6.30165767]]
```

If you had used an instance-based learning algorithm instead, you would have found that Israel has the closest GDP per capita to that of Cyprus ($38,341), and since the OECD data tells us that Israelis' life satisfaction is 7.2, you would have predicted a life satisfaction of 7.2 for Cyprus. If you zoom out a bit and look at the two next-closest countries, you will find Lithuania and Slovenia, both with a life satisfaction of 5.9. Averaging these three values, you get 6.33, which is pretty close to your model-based prediction. This simple algorithm is called *k-nearest neighbors* regression (in this example, $k = 3$).

Replacing the linear regression model with k-nearest neighbors regression in the previous code is as easy as replacing these lines:

```python
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

with these two:

```python
from sklearn.neighbors import KNeighborsRegressor
model = KNeighborsRegressor(n_neighbors=3)
```

MAIN CHALLENGES OF MACHINE LEARNING

- وقتی مدل train میشه میبینیم خوب عمل نکرده. به طور کلی این دو دلیل داره: مدلمون بد بوده – دیتامون بد بوده، اول بریم سراغ دومی:
- ۱.insufficient quantity of training data: به یه بچه بخوای یاد بدی سیب چیه، جلوش یه سیب میذاری ۵ بار بهش میگی این سیبه حالا هر سیب دیگه ای جلوش بذاری بپرسی اسمش چیه بهت میگه سیب. ولی کامپیوتر این شکلی نیست باید کلی دیتای مختلف ببینه و یادبگیره، پس باید دیتامون به اندازه کافی زیاد باشه.

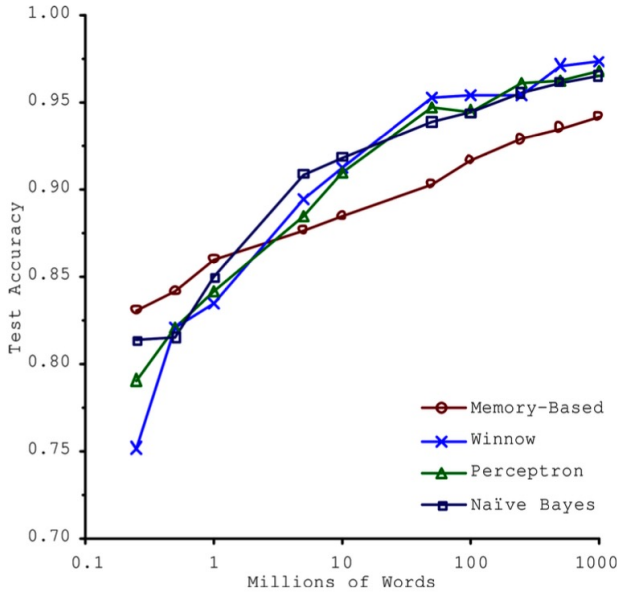

Figure 1-21. The importance of data versus algorithms[8]

- ۲.Nonrepresnetative training data: این مشکل وقتی به وجود میاد که دیتامون به اندازه کافی نمونه های خوبی از جامعه آماریمون نباشه. یعنی وقتی داریم کشور های مختلف رو با GDP های مختلف میاریم شامل همه GDP ها باشه و فقط یه سری کشور خاص داخل دیتاست نباشه و وقتی یه دیتا جدید بیاد و قبلا شبیهش رو داخل training set ندیده باشیم اون موقع خیلی پیشبینی دقیقی نخواهیم داشت.

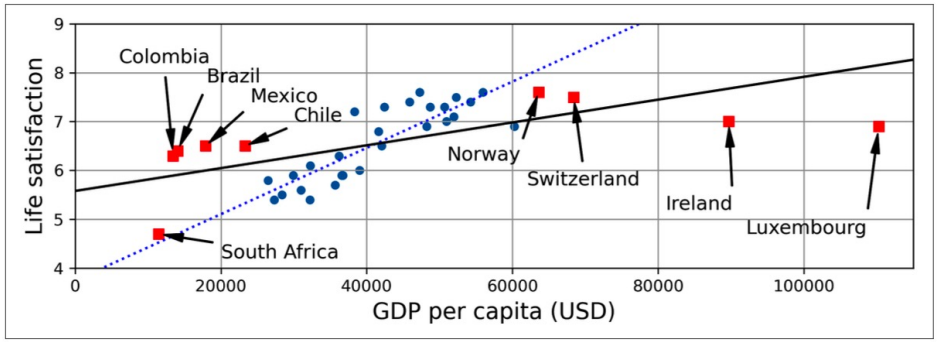

Figure 1-22. A more representative training sample

ادامه

- مفهوم sampling bias چیه؟ یه مثال خیلی معروف از این اتفاق برمیگرده به ۱۹۶۳ که اومدن نظرسنجی کردن قبل انتخابات امریکا و کارشون این بود ۱۰ میلیون نامه فرستادن و ۲.۶ میلیون نامه گرفتن و طبق اون رقیب روزولت میبرد ولی در نهایت تو انتخابات روزولت برنده شد. بخاطر اینکه سمپل هاشون بایاس داشته این نتجیه رخ داده. چطوری؟ اکثرا تو کلوب ها و جاهای پردرآمد اینارو فرستادن یا اینکه کلا ۲۵ درصد جواب دادن و بقیه ایگنور کردن که باعث شده صرفا ادمای رادیکال تر حاضر به جواب دهی بشن که شامل اکثریت قشر مخالف روزولت میشه.
- ۳. poor-quality data: وقتی که دیتامون داده پرت داخلش زیاد باشه این مشکل پیش میاد که نمیتونه یه مدل خوب رو یه سری دیتای منطقی باشه. اما داده پرت چیه؟ داده پرت داده ایه که خیلی نشون دهنده ویژگی های عمومی جامعه نیست و یه مورد خاصیه صرفا. یا اینکه حتی یه داده ای باشه که همه ویژگی هاشو نداشته باشیم. مثلا اگه داده یه سری ادم رو جمع کنیم و یه سریاشون سنشون رو ننوشته باشن. به طور کلی میصرفه وقت بذاریم و این دیتاها رو یا پاک کنیم یا درست کنیم تا نتیجه مدلمون بهتر بشه.
- ۴. irrelevant features: به طور کلی وقتی مدلمون میتونه پیش‌بینی خوبی داشته باشه که یه سری فیچر خوب بهش بدیم که با خروجی واقعا مرتبط باشه. برای مثال متراژ خونه خیلی بدرد بخور تر از طبقه واحد داخل قیمت خونس پس سعی میکنیم فیچر هایی رو پاس بدیم که بدرد بخور باشن بهش میگن feature engineering. یا بیایم یه سری فیچر رو با هم ترکیب کنیم که قبلا بهش اشاره کرده بودیم feature extraction.

ادامه

- ۵. overfitting the training data: ممکنه بری از یه مغازه خرید کنی و طرف تو پاچت کنه و بیای بگی همه همین شکلین به این میگن تعمیم کلی. ما ادم ها بعضی وقتا دچارش میشیم که همینم برای کامپیوتر ها ممکنه پیش بیاد که بهش میگن ovverfitting اما دقیقا یعنی چی؟ معنی دقیقش میشه که روی training set بیاد خیلی عملکرد خوبی داشته باشه مدلمون ولی روی یک سری دیتای جدید که بهش پاس میدیم پیش‌بینی کنه نتونه خوب عمل کنه. مثلا همون مدل تخمین خوشحالی کشور های مختلف میتونیم به جای اینکه از یه مدل خطی استفاده کنیم از یه مدل چند درجه ای بالا استفاده کنیم که باعث میشه خیلی خوب بتونه دیتاهای داخل training set رو براشون مدل رسم کنه ولی تو پیش بینی یه نمونه جدید خوب عمل نمیکنه.

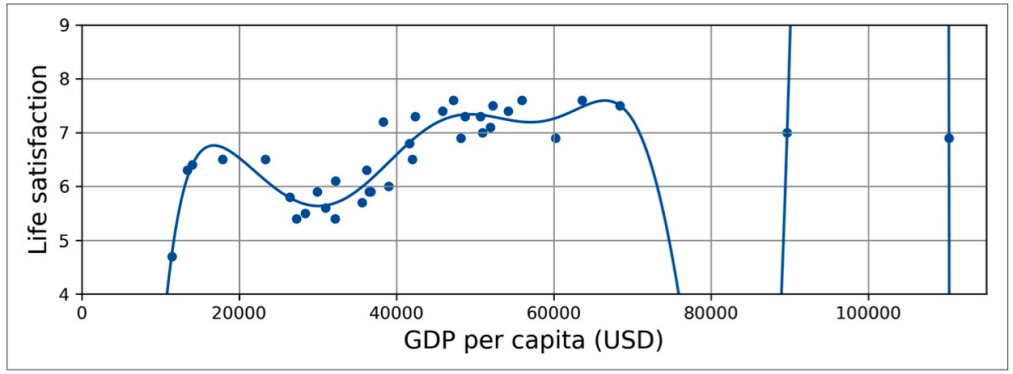

Figure 1-23. Overfitting the training data

درواقع وقتی میایم یه سری روابط خیلی پیچیده که واقعا ربطی به دیتاهامون نداره رو کشف میکنیم و مبنا تصمیم گیری میذاریم باعث همچین اتفاقی میشه. برای مثال اگه این کشور هارو به یه شبکه عصبی میدادیم ممکن بود بیاد بگه کشور هایی که اول اسمشون W داره کشورهایین که وضعشون خوبه پس این یه معیار. ولی وقتی بهش یه کشور ضعیف با W بدیم که یه نمونه جدیده اون موقع اشتباه میکنه. در واقع این پترنی که کشف کرده خیلی پیچیدس و اصلا وجود نداره و نباید مبنا قرار بگیره.

ادامه

- چطوری مشکل overfitting رو حل کنیم؟ بیایم تعداد پارامتر های مدل رو کم کنیم (مثلا به جای استفاده از یه مدل چند درجه ای از یه مدل خطی استفاده کنیم) یا اینکه دیتاهای بیشتری رو جمع اوری کنیم تا یه سری پترن اشتباه به خاطر بایاس داشتن دیتاهامون پیدا نکنه یا اینکه داده های پرت رو داخل training set رو حذف کنیم.
- ۶. underfitting: برعکس overfitting هستش. وقتی پیش میاد که مدل نه روی training set نه روی نمونه های جدید نمیتونه ارزیابی خوبی داشته باشه و دقتش پایینه. درواقع چون مدل ما بیش از حد سادس نمیتونه روابط پیچیده داخلی رو کشف کنه و باعث میشه نتونه پترن داخلی بین دیتا رو پیدا کنه.برای مثال اگه یه مسئله ماهیت چندجمله ای داشته باشه و ما بخوایم با یه مدل خطی بریم جلو به این مشکل برمیخوریم. راه حل؟ مدل پیچیده تر انتخاب کنیم، پارامتر های بیشتری پاس بدیم تا یادبگیره.

TESTING AND VALIDATING

- برای اینکه بفهمیم مدل دقیقا چقدر خوب رو یه سری دیتای جدید میتونه عمل کنه. میتونیم مدل رو بذاریم رو برنامه اصلی و وایسیم تا عملکردش رو ببینیم ولی این خوب نیست. میتونیم یه کار دیگه بکنیم و اونم اینه training set رو به دو بخش تقسیم کنیم . Test set و train set. از train set برای اینکه مدلمون رو بهش یاد بدیم استفاده میکنیم و بعد از اینکه کارش تموم شد برای اینکه بفهمیم چقدر خطا داره و اینکه چقدر خوب روی یه سری دیتا که اصلا ندیده عمل میکنه میتونیم از test set استفاده کنیم. اگه training error که همون اروریه که از پیش‌بینی خروجی مدلمون روی دیتاهای داخل train set محاسبه میشه، کم باشه. ولی test error زیاد باشه میفهمیم overfitting پیش اومده.

It is common to use 80% of the data for training and *hold out* 20% for testing. However, this depends on the size of the dataset: if it contains 10 million instances, then holding out 1% means your test set will contain 100,000 instances, probably more than enough to get a good estimate of the generalization error.

HYPERPARAMETER TUNING AND MODEL SELECTION

- فرض کنید که میخوایم از بین دو مدل انتخاب کنیم کدوم تازه ممکنه هر کدوم از این دو مدل کلی hyperparameter داشته باشن که باعث میشه یه مدل رو کلی به حالت های مختلف بخوایم train کنیم. کدوم رو انتخاب کنیم؟ بیایم برای یه مدل واقعا ۱۰۰ تا حالت مختلف رو امتحان کنیم و نتیجه رو ببینیم چی میشه؟ این روش حتی یه مشکل داره، ممکنه تو بری ۱۰۰ تا مدل رو یه الگوریتم با hyperparameter های مختلف train کنی بعد بهترین رو ببری روی production و ببینی عه اورش بیشتر از چیزی بود که قبلا باهاش اندازه گیری کرده بودی تا بهترین رو انتخاب کنی.(به خاطر اینکه مدل برای بار های زیادی تست ست رو دیده خودشو جوری طبیق داده که ارور اون رو کم کنه، در حالی که تعمیم پذیریش خراب شده) چی کار میشه کرد؟
- استفاده از holdout validation. کاری که میکینم اینه train set رو به دو بخش train و dev تقسیم میکنیم. میایم مدل های مختلف رو بر اساس بخش باقی مونده train میکنیم و سپس با hyperparameter، dev set هاشو اپدیت میکنیم و میفهمیم کدومشون از همه بهتره و یه دور رو کل training set، trainش میکینم و در نهایت ارور کلی اون مدل رو با test set برسی میکنیم.

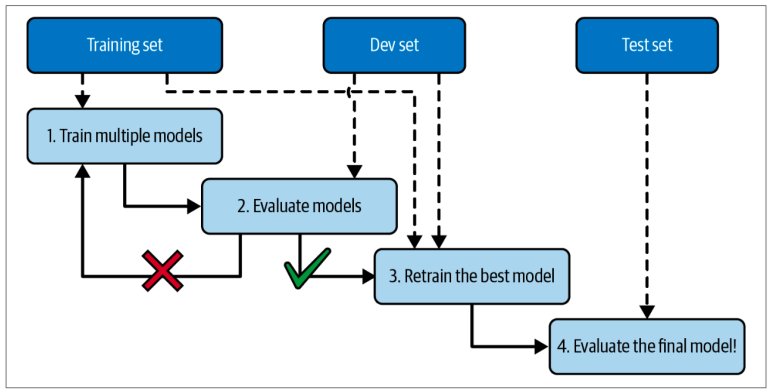

Figure 1-25. Model selection using holdout validation

Problems with Using Only Training and Test Sets

1. **Hyperparameter Tuning and Model Selection**:
   - **Without Validation Set**: When you use only a training set and a test set, you might be tempted to use the test set to tune hyperparameters and select the best model. This can lead to overfitting on the test set because you are indirectly optimizing the model based on this set.
   - **Problem**: The test set, which should provide an unbiased evaluation of the final model's performance, becomes compromised. The performance metrics obtained from the test set may not accurately reflect the model's true generalization ability to unseen data.

CROSS VALIDATION

- روش قبلی اوکیه ولی اگه dev set خیلی کوچیک باشه معیار خوبی نخواهد بود یا برعکس اگه خیلی بزرگ باشه هم خوب نیست چون اون موقع train set کوچیک میشه و اینم معیار خوبی نیست. پس کاری که میکنیم اینه که بیایم cross validation بزنیم. چی کار میکنه؟ کلی dev set های مختلف هر سری از جاهای مختلف training set برمیداره و با اون میاد برسی میکنه دقت مدل رو و در نهایت پرفورمنس هر مدل رو میاد بر اساس میانگین تمامی این ارور ها که هر کدوم متعلق به یه dev set هست برسی میکنه. بدی این کار اینه صرفا تایم train کردن و انتخاب کردن بهترین مدل بیشتر میشه.

- اما data mismatch چیه؟فرض کنید که میخوایم یه برنامه تشخیص گل بسازیم و نکته خوب اینه میتونیم کلی عکس از اینترنت دانلود کنیم اما نکته ای که وجود داره اینه وقتی ببریم رو پروداکشن میبینیم برناممون داره ضعیف عمل میکنه چرا؟ چون عموما از یه سری گل خاص احتمالا قراره تشخیص بده داخل برناممون. پس باید داخل test set و dev set اون شکل خاص از گل که قراره کلی استفاده بشه داخل اینا قرار بدیم. در واقع کاری که میکنیم اینه داخل این دو مجموعه نمونه ای رو قرار میدیم که احتمال میدیم قراره داخل پروداکشن باهاش برخورد کنیم. اما اگه به یه پرفورمنس بد برخورد کنیم وقتی روی dev set برخورد کردیم از کجا بدونیم مشکل overfitting یا data mismatch هستش؟ یه راه اینه بیایم train-dev درست کنیم و با این معیار اول بفهمیم کدوم مدل روی کل گل هایی که از اینترنت دانلود کردیم عملکرد بهتری داره و overfitting براش پیش نیومده . و اگه همچین مدلی روی dev set عملکردش ضعیف باشه میفهمم data mismatch رخ داده.

DATA MISS MATCH

One solution is to hold out some of the training pictures (from the web) in yet another set that Andrew Ng dubbed the *train-dev set* (Figure 1-26). After the model is trained (on the training set, *not* on the train-dev set), you can evaluate it on the train-dev set. If the model performs poorly, then it must have overfit the training set, so you should try to simplify or regularize the model, get more training data, and clean up the training data. But if it performs well on the train-dev set, then you can evaluate the model on the dev set. If it performs poorly, then the problem must be coming from the data mismatch. You can try to tackle this problem by preprocessing the web images to make them look more like the pictures that will be taken by the mobile app, and then retraining the model. Once you have a model that performs well on both the train-dev set and the dev set, you can evaluate it one last time on the test set to know how well it is likely to perform in production.

Figure 1-26. When real data is scarce (right), you may use similar abundant data (left) for training and hold out some of it in a train-dev set to evaluate overfitting; the real data is then used to evaluate data mismatch (dev set) and to evaluate the final model's performance (test set)

NO FREE LUNCH THEOREM

No Free Lunch Theorem

A model is a simplified representation of the data. The simplifications are meant to discard the superfluous details that are unlikely to generalize to new instances. When you select a particular type of model, you are implicitly making *assumptions* about the data. For example, if you choose a linear model, you are implicitly assuming that the data is fundamentally linear and that the distance between the instances and the straight line is just noise, which can safely be ignored.

In a famous 1996 paper,[9] David Wolpert demonstrated that if you make absolutely no assumption about the data, then there is no reason to prefer one model over any other. This is called the *No Free Lunch* (NFL) theorem. For some datasets the best model is a linear model, while for other datasets it is a neural network. There is no model that is *a priori* guaranteed to work better (hence the name of the theorem). The only way to know for sure which model is best is to evaluate them all. Since this is not possible, in practice you make some reasonable assumptions about the data and evaluate only a few reasonable models. For example, for simple tasks you may evaluate linear models with various levels of regularization, and for a complex problem you may evaluate various neural networks.