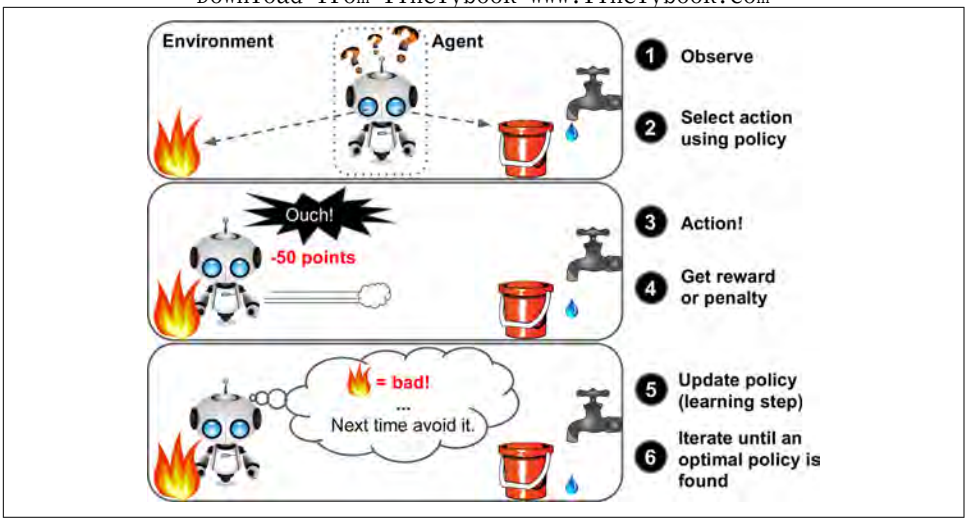

Figure 1-12. Reinforcement Learning

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 March 2016 when it beat the world champion Lee Sedol 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.

Batch and Online Learning

Another criterion used to classify Machine Learning systems is whether or not the system can learn incrementally from a stream of incoming data.

Batch learning

In *batch learning*, the system is incapable of learning incrementally: it must be trained using all the available data. This will generally take a lot of time and computing resources, so it is typically done offline. First the system is trained, and then it is launched into production and runs without learning anymore; it just applies what it has learned. This is called *offline learning*.

If you want a batch learning system to know about new data (such as a new type of spam), you need to train a new version of the system from scratch on the full dataset (not just the new data, but also the old data), then stop the old system and replace it with the new one.

Fortunately, the whole process of training, evaluating, and launching a Machine Learning system can be automated fairly easily (as shown in Figure 1-3), so even a

batch learning system can adapt to change. Simply update the data and train a new version of the system from scratch as often as needed.

This solution is simple and often works fine, but training using the full set of data can take many hours, so you would typically train a new system only every 24 hours or even just weekly. If your system needs to adapt to rapidly changing data (e.g., to predict stock prices), then you need a more reactive solution.

Also, training on the full set of data requires a lot of computing resources (CPU, memory space, disk space, disk I/O, network I/O, etc.). If you have a lot of data and you automate your system to train from scratch every day, it will end up costing you a lot of money. If the amount of data is huge, it may even be impossible to use a batch learning algorithm.

Finally, if your system needs to be able to learn autonomously and it has limited resources (e.g., a smartphone application or a rover on Mars), then carrying around large amounts of training data and taking up a lot of resources to train for hours every day is a showstopper.

Fortunately, a better option in all these cases is to use algorithms that are capable of learning incrementally.

Online learning

In *online learning*, you train the system incrementally by feeding it data instances sequentially, either individually or by small groups called *mini-batches*. Each learning step is fast and cheap, so the system can learn about new data on the fly, as it arrives (see Figure 1-13).

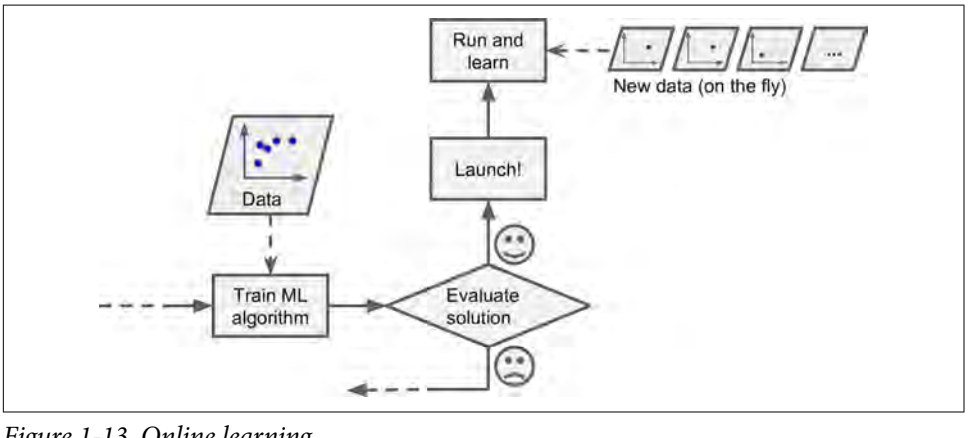

Figure 1-13. Online learning

Online learning is great for systems that receive data as a continuous flow (e.g., stock prices) and need to adapt to change rapidly or autonomously. It is also a good option

if you have limited computing resources: once an online learning system has learned about new data instances, it does not need them anymore, so you can discard them (unless you want to be able to roll back to a previous state and "replay" the data). This can save a huge amount of space.

Online learning algorithms can also be used to train systems on huge datasets that cannot fit in one machine's main memory (this is called *out-of-core* learning). The algorithm loads part of the data, runs a training step on that data, and repeats the process until it has run on all of the data (see Figure 1-14).

This whole process is usually done offline (i.e., not on the live system), so *online learning* can be a confusing name. Think of it as *incremental learning*.

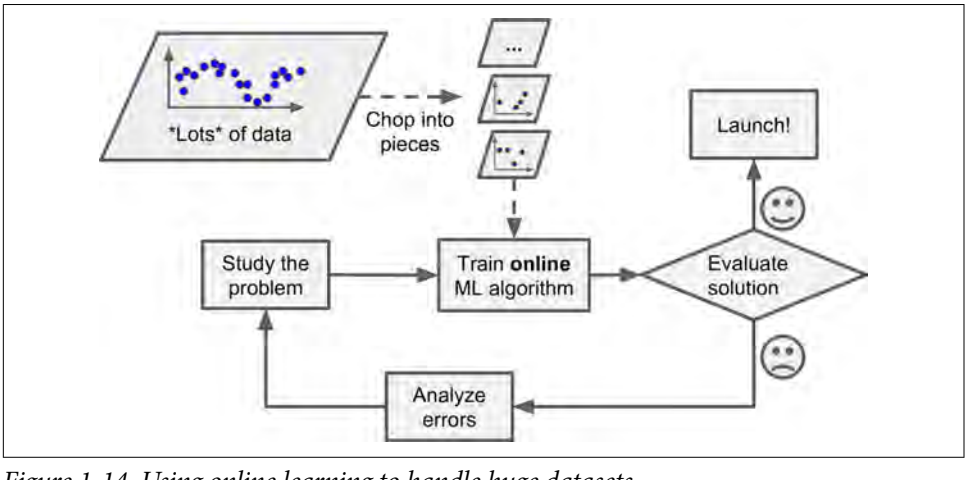

Figure 1-14. Using online learning to handle huge datasets

One important parameter of online learning systems is how fast they should adapt to changing data: this is called the *learning rate*. If you set a high learning rate, then your system will rapidly adapt to new data, but it will also tend to quickly forget the old data (you don't want a spam filter to flag only the latest kinds of spam it was shown). Conversely, if you set a low learning rate, the system will have more inertia; that is, it will learn more slowly, but it will also be less sensitive to noise in the new data or to sequences of nonrepresentative data points.

A big challenge with online learning is that if bad data is fed to the system, the system's performance will gradually decline. If we are talking about a live system, your clients will notice. For example, bad data could come from a malfunctioning sensor on a robot, or from someone spamming a search engine to try to rank high in search

results. To reduce this risk, you need to monitor your system closely and promptly switch learning off (and possibly revert to a previously working state) if you detect a drop in performance. You may also want to monitor the input data and react to abnormal data (e.g., using an anomaly detection algorithm).

Instance-Based Versus Model-Based Learning

One more way to categorize Machine Learning systems is by how they *generalize*. Most Machine Learning tasks are about making predictions. This means that given a number of training examples, the system needs to be able to generalize to examples it has never seen before. Having a good performance measure on the training data is good, but insufficient; the true goal is to perform well on new instances.

There are two main approaches to generalization: instance-based learning and model-based learning.

Instance-based learning

Possibly the most trivial form of learning is simply to learn by heart. If you were to create a spam filter this way, it would just flag all emails that are identical to emails that have already been flagged by users—not the worst solution, but certainly not the best.

Instead of just flagging emails that are identical to known spam emails, your spam filter could be programmed to also flag emails that are very similar to known spam emails. This requires a *measure of similarity* between two emails. A (very basic) similarity measure between two emails could be to count the number of words they have in common. The system would flag an email as spam if it has many words in common with a known spam email.

This is called *instance-based learning*: the system learns the examples by heart, then generalizes to new cases using a similarity measure (Figure 1-15).

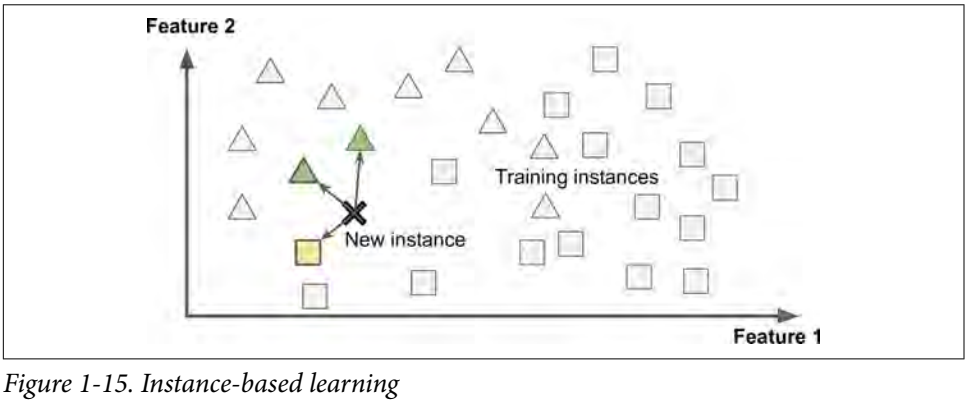

Figure 1-15. Instance-based learning