

---

# The Machine Learning Landscape

Not so long ago, if you had picked up your phone and asked it the way home, it would have ignored you—and people would have questioned your sanity. But machine learning is no longer science fiction: billions of people use it every day. And the truth is it has actually been around for decades in some specialized applications, such as optical character recognition (OCR). The first ML application that really became mainstream, improving the lives of hundreds of millions of people, took over the world back in the 1990s: the *spam filter*. It's not exactly a self-aware robot, but it does technically qualify as machine learning: it has actually learned so well that you seldom need to flag an email as spam anymore. It was followed by hundreds of ML applications that now quietly power hundreds of products and features that you use regularly: voice prompts, automatic translation, image search, product recommendations, and many more.

Where does machine learning start and where does it end? What exactly does it mean for a machine to *learn* something? If I download a copy of all Wikipedia articles, has my computer really learned something? Is it suddenly smarter? In this chapter I will start by clarifying what machine learning is and why you may want to use it.

Then, before we set out to explore the machine learning continent, we will take a look at the map and learn about the main regions and the most notable landmarks: supervised versus unsupervised learning and their variants, online versus batch learning, instance-based versus model-based learning. Then we will look at the workflow of a typical ML project, discuss the main challenges you may face, and cover how to evaluate and fine-tune a machine learning system.

This chapter introduces a lot of fundamental concepts (and jargon) that every data scientist should know by heart. It will be a high-level overview (it's the only chapter without much code), all rather simple, but my goal is to ensure everything is crystal

clear to you before we continue on to the rest of the book. So grab a coffee and let's get started!



If you are already familiar with machine learning basics, you may want to skip directly to [Chapter 2](#). If you are not sure, try to answer all the questions listed at the end of the chapter before moving on.

## What Is Machine Learning?

Machine learning is the science (and art) of programming computers so they can *learn from data*.

Here is a slightly more general definition:

[Machine learning is the] field of study that gives computers the ability to learn without being explicitly programmed.

—Arthur Samuel, 1959

And a more engineering-oriented one:

A computer program is said to learn from experience  $E$  with respect to some task  $T$  and some performance measure  $P$ , if its performance on  $T$ , as measured by  $P$ , improves with experience  $E$ .

—Tom Mitchell, 1997

Your spam filter is a machine learning program that, given examples of spam emails (flagged by users) and examples of regular emails (nonspam, also called “ham”), can learn to flag spam. The examples that the system uses to learn are called the *training set*. Each training example is called a *training instance* (or *sample*). The part of a machine learning system that learns and makes predictions is called a *model*. Neural networks and random forests are examples of models.

In this case, the task  $T$  is to flag spam for new emails, the experience  $E$  is the *training data*, and the performance measure  $P$  needs to be defined; for example, you can use the ratio of correctly classified emails. This particular performance measure is called *accuracy*, and it is often used in classification tasks.

If you just download a copy of all Wikipedia articles, your computer has a lot more data, but it is not suddenly better at any task. This is not machine learning.



# Why Use Machine Learning?

Consider how you would write a spam filter using traditional programming techniques (Figure 1-1):

1. First you would examine what spam typically looks like. You might notice that some words or phrases (such as “4U”, “credit card”, “free”, and “amazing”) tend to come up a lot in the subject line. Perhaps you would also notice a few other patterns in the sender’s name, the email’s body, and other parts of the email.
2. You would write a detection algorithm for each of the patterns that you noticed, and your program would flag emails as spam if a number of these patterns were detected.
3. You would test your program and repeat steps 1 and 2 until it was good enough to launch.

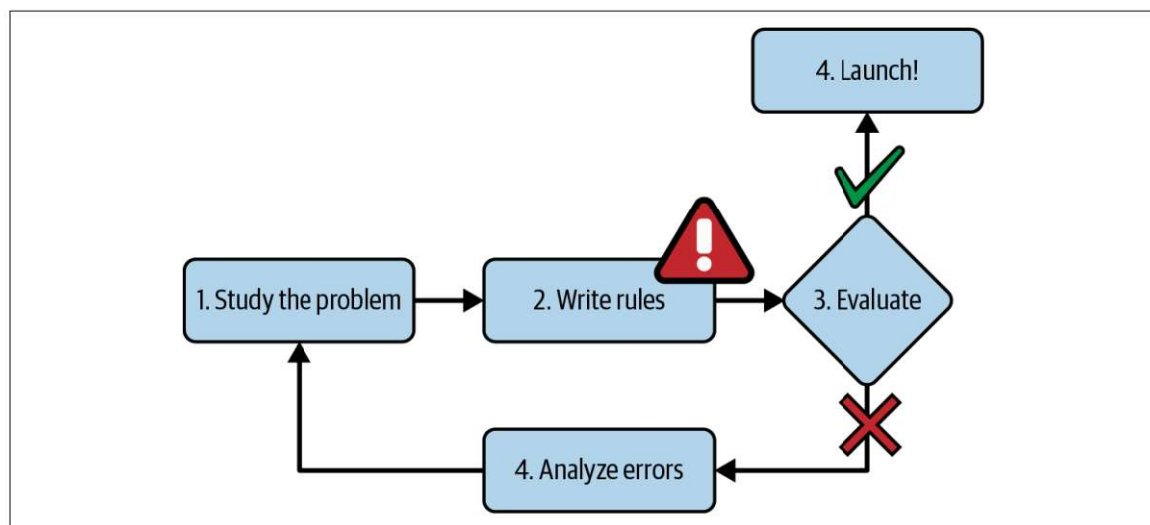


Figure 1-1. *The traditional approach*

Since the problem is difficult, your program will likely become a long list of complex rules—pretty hard to maintain.

In contrast, a spam filter based on machine learning techniques automatically learns which words and phrases are good predictors of spam by detecting unusually frequent patterns of words in the spam examples compared to the ham examples (Figure 1-2). The program is much shorter, easier to maintain, and most likely more accurate.

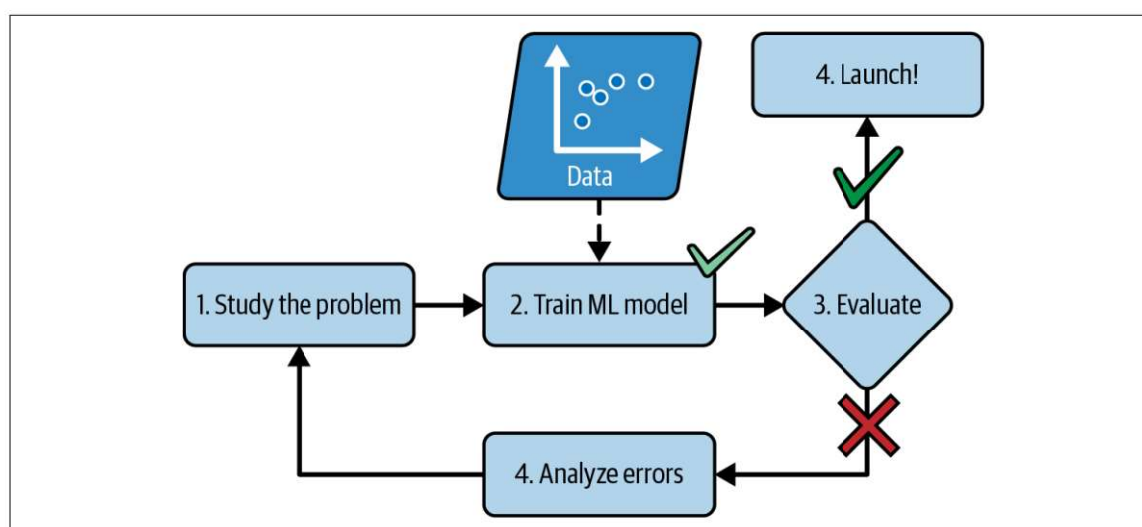


Figure 1-2. *The machine learning approach*

What if spammers notice that all their emails containing “4U” are blocked? They might start writing “For U” instead. A spam filter using traditional programming techniques would need to be updated to flag “For U” emails. If spammers keep working around your spam filter, you will need to keep writing new rules forever.

In contrast, a spam filter based on machine learning techniques automatically notices that “For U” has become unusually frequent in spam flagged by users, and it starts flagging them without your intervention (Figure 1-3).

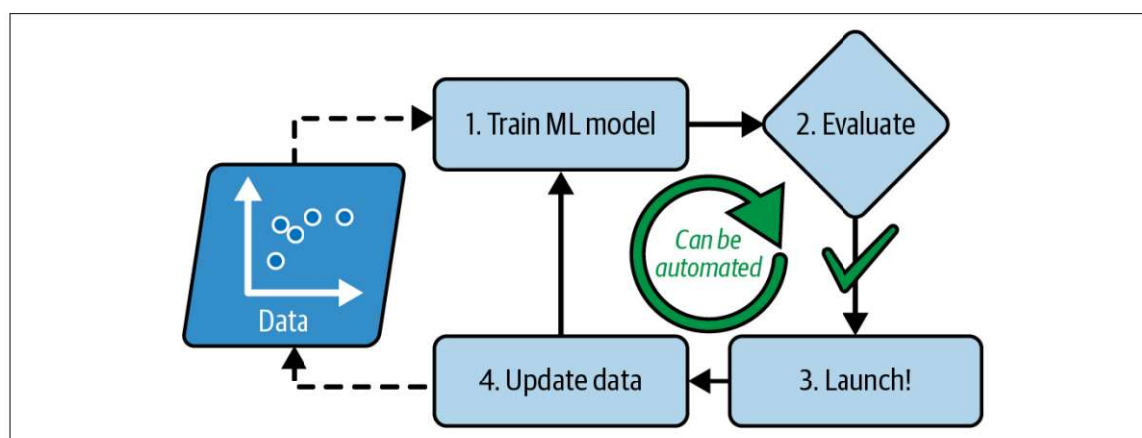


Figure 1-3. *Automatically adapting to change*

Another area where machine learning shines is for problems that either are too complex for traditional approaches or have no known algorithm. For example, consider speech recognition. Say you want to start simple and write a program capable of distinguishing the words “one” and “two”. You might notice that the word “two” starts with a high-pitch sound (“T”), so you could hardcode an algorithm that measures high-pitch sound intensity and use that to distinguish ones and twos—but obviously



this technique will not scale to thousands of words spoken by millions of very different people in noisy environments and in dozens of languages. The best solution (at least today) is to write an algorithm that learns by itself, given many example recordings for each word.

Finally, machine learning can help humans learn (Figure 1-4). ML models can be inspected to see what they have learned (although for some models this can be tricky). For instance, once a spam filter has been trained on enough spam, it can easily be inspected to reveal the list of words and combinations of words that it believes are the best predictors of spam. Sometimes this will reveal unsuspected correlations or new trends, and thereby lead to a better understanding of the problem. Digging into large amounts of data to discover hidden patterns is called *data mining*, and machine learning excels at it.

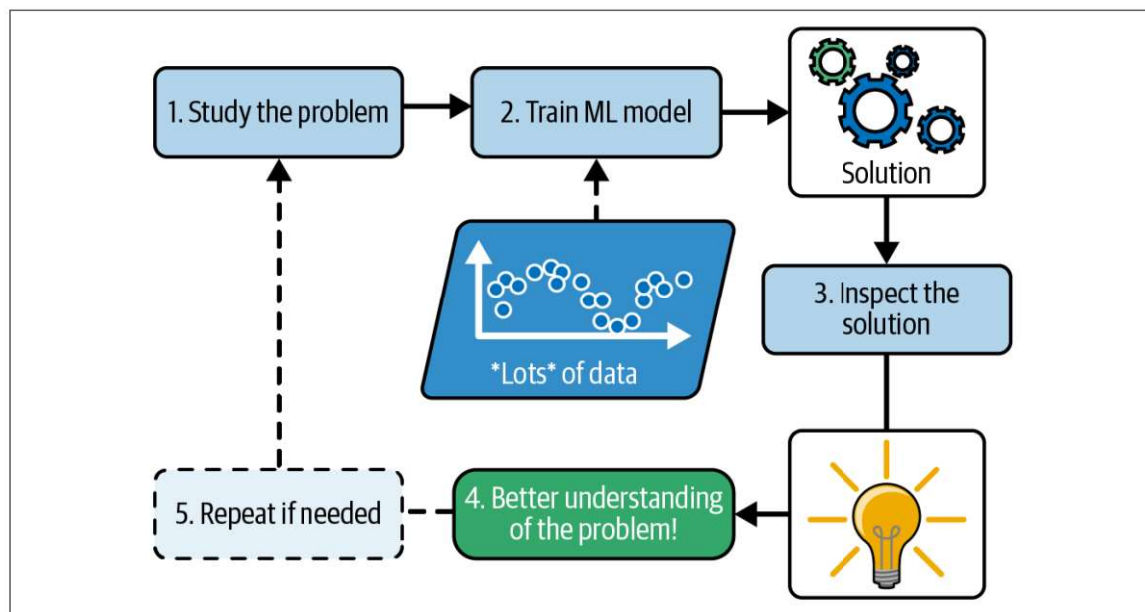


Figure 1-4. Machine learning can help humans learn

To summarize, machine learning is great for:

- Problems for which existing solutions require a lot of fine-tuning or long lists of rules (a machine learning model can often simplify code and perform better than the traditional approach)
- Complex problems for which using a traditional approach yields no good solution (the best machine learning techniques can perhaps find a solution)
- Fluctuating environments (a machine learning system can easily be retrained on new data, always keeping it up to date)
- Getting insights about complex problems and large amounts of data

# Examples of Applications

Let's look at some concrete examples of machine learning tasks, along with the techniques that can tackle them:

*Analyzing images of products on a production line to automatically classify them*

This is image classification, typically performed using convolutional neural networks (CNNs; see [Chapter 14](#)) or sometimes transformers (see [Chapter 16](#)).

*Detecting tumors in brain scans*

This is semantic image segmentation, where each pixel in the image is classified (as we want to determine the exact location and shape of tumors), typically using CNNs or transformers.

*Automatically classifying news articles*

This is natural language processing (NLP), and more specifically text classification, which can be tackled using recurrent neural networks (RNNs) and CNNs, but transformers work even better (see [Chapter 16](#)).

*Automatically flagging offensive comments on discussion forums*

This is also text classification, using the same NLP tools.

*Summarizing long documents automatically*

This is a branch of NLP called text summarization, again using the same tools.

*Creating a chatbot or a personal assistant*

This involves many NLP components, including natural language understanding (NLU) and question-answering modules.

*Forecasting your company's revenue next year, based on many performance metrics*

This is a regression task (i.e., predicting values) that may be tackled using any regression model, such as a linear regression or polynomial regression model (see [Chapter 4](#)), a regression support vector machine (see [Chapter 5](#)), a regression random forest (see [Chapter 7](#)), or an artificial neural network (see [Chapter 10](#)). If you want to take into account sequences of past performance metrics, you may want to use RNNs, CNNs, or transformers (see [Chapters 15 and 16](#)).

*Making your app react to voice commands*

This is speech recognition, which requires processing audio samples: since they are long and complex sequences, they are typically processed using RNNs, CNNs, or transformers (see [Chapters 15 and 16](#)).

*Detecting credit card fraud*

This is anomaly detection, which can be tackled using isolation forests, Gaussian mixture models (see [Chapter 9](#)), or autoencoders (see [Chapter 17](#)).



*Segmenting clients based on their purchases so that you can design a different marketing strategy for each segment*

This is clustering, which can be achieved using *k*-means, DBSCAN, and more (see [Chapter 9](#)).

*Representing a complex, high-dimensional dataset in a clear and insightful diagram*

This is data visualization, often involving dimensionality reduction techniques (see [Chapter 8](#)).

*Recommending a product that a client may be interested in, based on past purchases*

This is a recommender system. One approach is to feed past purchases (and other information about the client) to an artificial neural network (see [Chapter 10](#)), and get it to output the most likely next purchase. This neural net would typically be trained on past sequences of purchases across all clients.

*Building an intelligent bot for a game*

This is often tackled using reinforcement learning (RL; see [Chapter 18](#)), which is a branch of machine learning that trains agents (such as bots) to pick the actions that will maximize their rewards over time (e.g., a bot may get a reward every time the player loses some life points), within a given environment (such as the game). The famous AlphaGo program that beat the world champion at the game of Go was built using RL.

This list could go on and on, but hopefully it gives you a sense of the incredible breadth and complexity of the tasks that machine learning can tackle, and the types of techniques that you would use for each task.

## Types of Machine Learning Systems

There are so many different types of machine learning systems that it is useful to classify them in broad categories, based on the following criteria:

- How they are supervised during training ([supervised](#), [unsupervised](#), [semi-supervised](#), [self-supervised](#), and others)
- Whether or not they can learn incrementally on the fly (online versus batch learning)
- Whether they work by simply comparing new data points to known data points, or instead by detecting patterns in the training data and building a predictive model, much like scientists do (instance-based versus model-based learning)

These criteria are not exclusive; you can combine them in any way you like. For example, a state-of-the-art spam filter may learn on the fly using a deep neural network model trained using human-provided examples of spam and ham; this makes it an online, model-based, supervised learning system.

Let's look at each of these criteria a bit more closely.

## Training Supervision

ML systems can be classified according to the amount and type of supervision they get during training. There are many categories, but we'll discuss the main ones: supervised learning, unsupervised learning, self-supervised learning, semi-supervised learning, and reinforcement learning.

### Supervised learning

In *supervised learning*, the **training set** you feed to the algorithm **includes** the desired **solutions**, called **labels** (Figure 1-5).

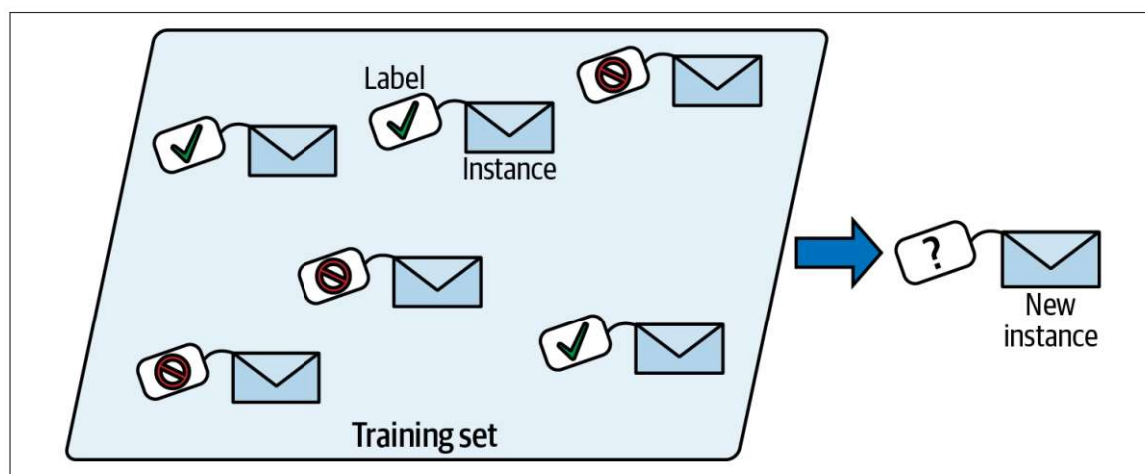


Figure 1-5. A labeled training set for spam classification (an example of supervised learning)

A typical supervised learning task is *classification*. The spam filter is a good example of this: it is trained with many example emails along with their *class* (spam or ham), and it must learn how to classify new emails.

Another typical task is to predict a *target* numeric value, such as the price of a car, given a set of *features* (mileage, age, brand, etc.). This sort of task is called *regression* (Figure 1-6).<sup>1</sup> To train the system, you need to give it many examples of cars, including both their features and their targets (i.e., their prices).

Note that some regression models can be used for classification as well, and vice versa. For example, *logistic regression* is commonly used for classification, as it can

<sup>1</sup> Fun fact: this odd-sounding name is a statistics term introduced by Francis Galton while he was studying the fact that the children of tall people tend to be shorter than their parents. Since the children were shorter, he called this *regression to the mean*. This name was then applied to the methods he used to analyze correlations between variables.



output a value that corresponds to the probability of belonging to a given class (e.g., 20% chance of being spam).

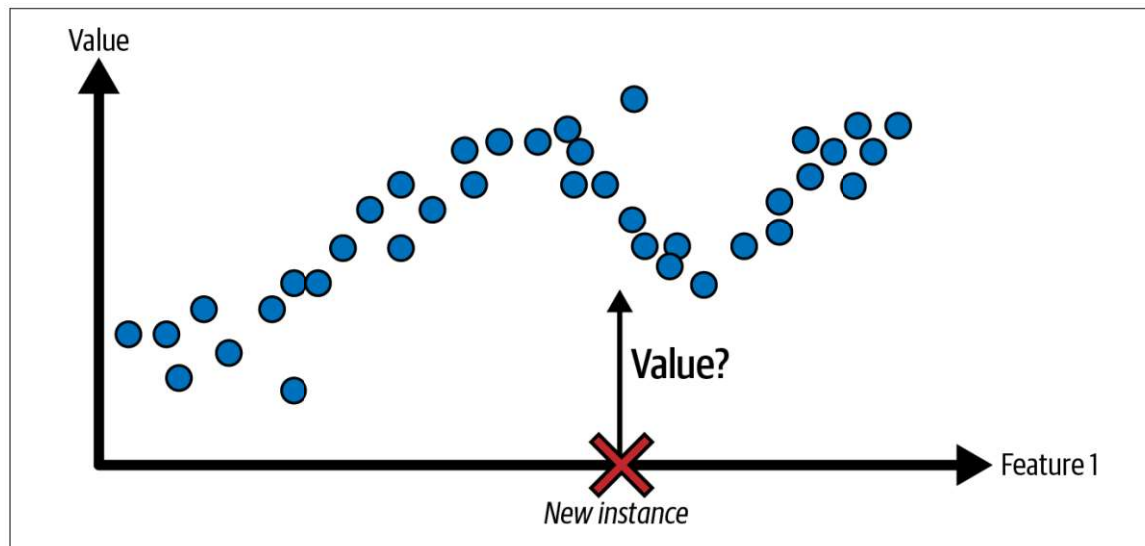


Figure 1-6. A regression problem: predict a value, given an input feature (there are usually multiple input features, and sometimes multiple output values)



The words *target* and *label* are generally treated as synonyms in supervised learning, but *target* is more common in regression tasks and *label* is more common in classification tasks. Moreover, *features* are sometimes called *predictors* or *attributes*. These terms may refer to individual samples (e.g., “this car’s mileage feature is equal to 15,000”) or to all samples (e.g., “the mileage feature is strongly correlated with price”).

## Unsupervised learning

In *unsupervised learning*, as you might guess, the training data is unlabeled (Figure 1-7). The system tries to learn without a teacher.

For example, say you have a lot of data about your blog’s visitors. You may want to run a *clustering* algorithm to try to detect groups of similar visitors (Figure 1-8). 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 use a *hierarchical clustering* algorithm, it may also subdivide each group into smaller groups. This may help you target your posts for each group.

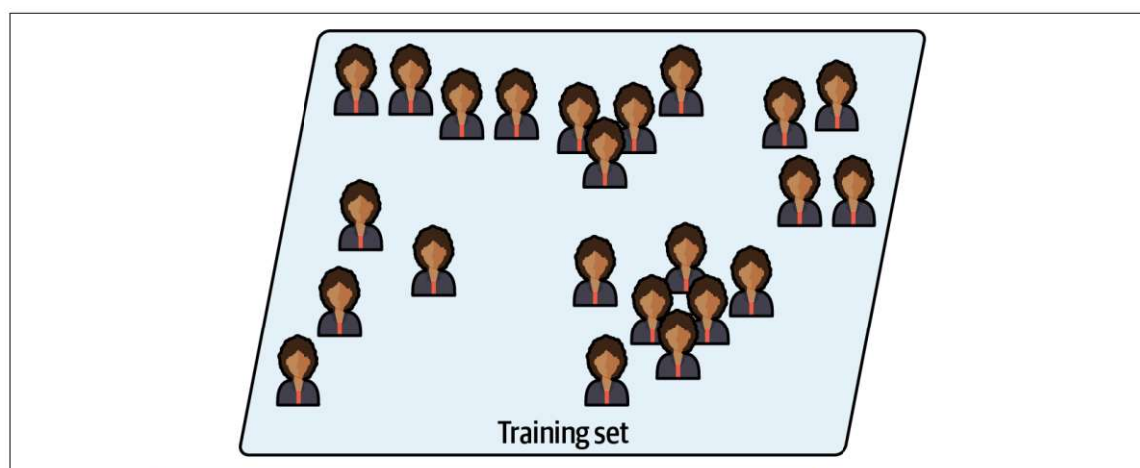


Figure 1-7. An unlabeled training set for unsupervised learning

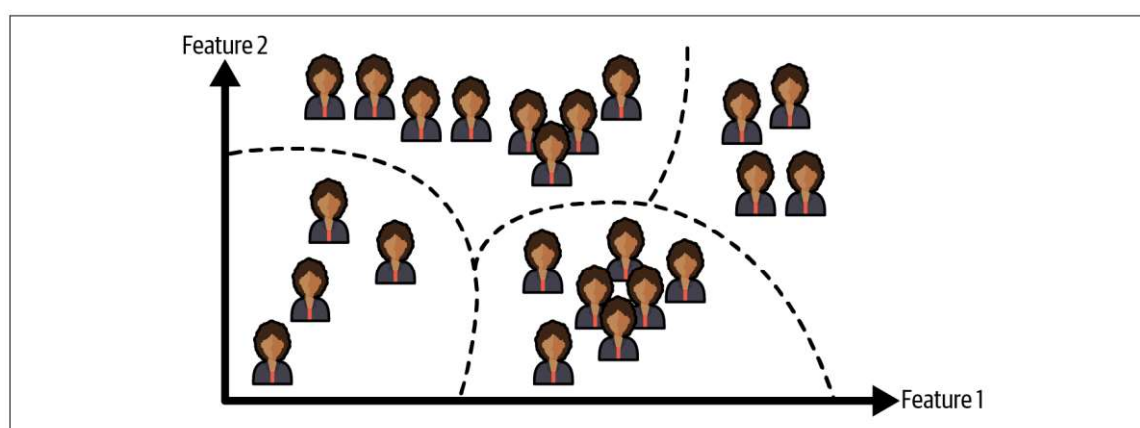


Figure 1-8. Clustering

*Visualization* algorithms are also good examples of unsupervised learning: you feed them a lot of complex and unlabeled data, and they output a 2D or 3D representation of your data that can easily be plotted (Figure 1-9). These algorithms try to preserve as much structure as they can (e.g., trying to keep separate clusters in the input space from overlapping in the visualization) so that you can understand how the data is organized and perhaps identify unsuspected patterns.

A related task is *dimensionality reduction*, in which the goal is to simplify the data without losing too much information. One way to do this is to merge several correlated features into one. For example, a car's mileage may be strongly correlated with its age, so the dimensionality reduction algorithm will merge them into one feature that represents the car's wear and tear. This is called *feature extraction*.



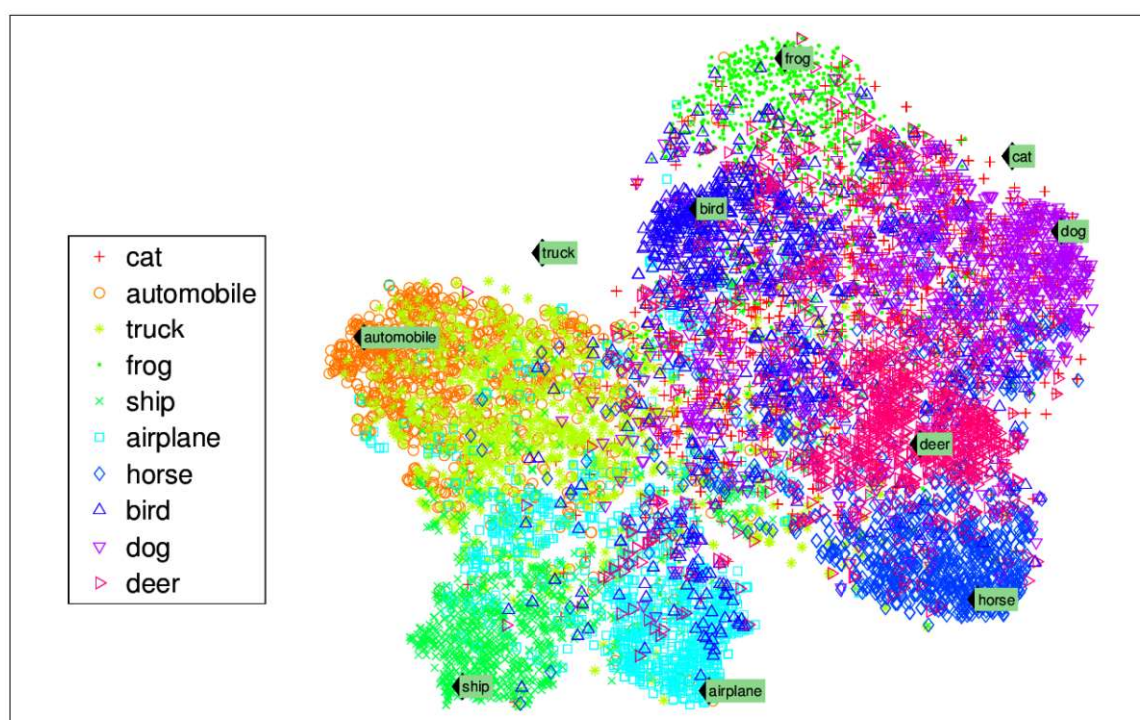


Figure 1-9. Example of a t-SNE visualization highlighting semantic clusters<sup>2</sup>



It is often a good idea to try to reduce the number of dimensions in your training data using a dimensionality reduction algorithm before you feed it to another machine learning algorithm (such as a supervised learning algorithm). It will run much faster, the data will take up less disk and memory space, and in some cases it may also perform better.

Yet another important unsupervised task is *anomaly detection*—for example, detecting unusual credit card transactions to prevent fraud, catching manufacturing defects, or automatically removing outliers from a dataset before feeding it to another learning algorithm. The system is shown mostly normal instances during training, so it learns to recognize them; then, when it sees a new instance, it can tell whether it looks like a normal one or whether it is likely an anomaly (see Figure 1-10). A very similar task is *novelty detection*: it aims to detect new instances that look different from all instances in the training set. This requires having a very “clean” training set, devoid of any instance that you would like the algorithm to detect. For example, if you have thousands of pictures of dogs, and 1% of these pictures represent Chihuahuas, then a

<sup>2</sup> Notice how animals are rather well separated from vehicles and how horses are close to deer but far from birds. Figure reproduced with permission from Richard Socher et al., “Zero-Shot Learning Through Cross-Modal Transfer”, *Proceedings of the 26th International Conference on Neural Information Processing Systems* 1 (2013): 935–943.

novelty detection algorithm should not treat new pictures of Chihuahuas as novelties. On the other hand, anomaly detection algorithms may consider these dogs as so rare and so different from other dogs that they would likely classify them as anomalies (no offense to Chihuahuas).

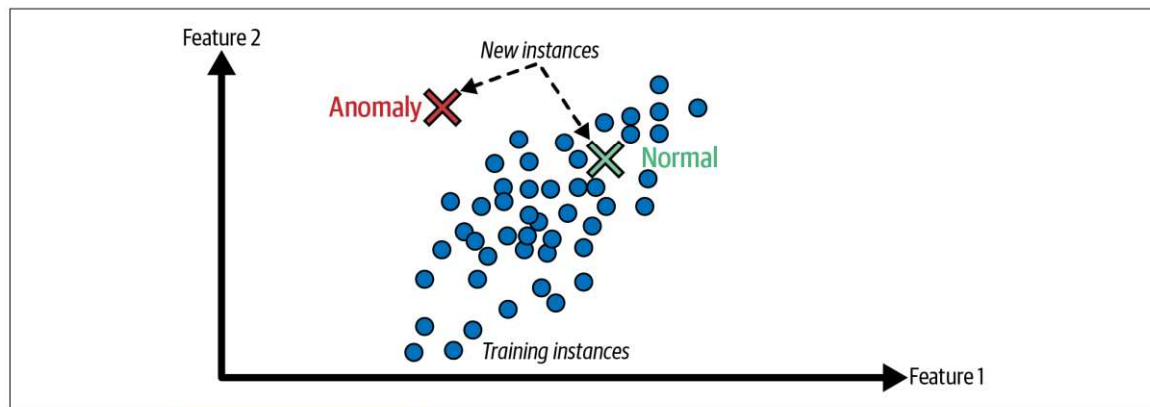


Figure 1-10. Anomaly detection

Finally, another common unsupervised task is *association rule learning*, in which the goal is to dig into large amounts of data and discover interesting relations between attributes. For example, suppose you own a supermarket. Running an association rule on your sales logs may reveal that people who purchase barbecue sauce and potato chips also tend to buy steak. Thus, you may want to place these items close to one another.

### Semi-supervised learning

Since labeling data is usually time-consuming and costly, you will often have plenty of unlabeled instances, and few labeled instances. Some algorithms can deal with data that's partially labeled. This is called *semi-supervised learning* (Figure 1-11).

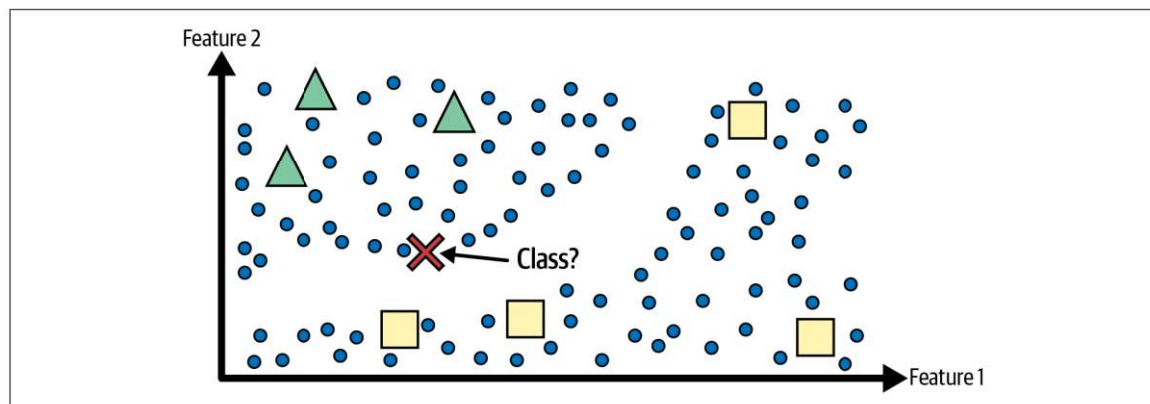


Figure 1-11. Semi-supervised learning with two classes (triangles and squares): the unlabeled examples (circles) help classify a new instance (the cross) into the triangle class rather than the square class, even though it is closer to the labeled squares



Some photo-hosting services, such as Google Photos, are good examples of this. Once you upload all your family photos to the service, it automatically recognizes that the same person A shows up in photos 1, 5, and 11, while another person B shows up in photos 2, 5, and 7. This is the unsupervised part of the algorithm (clustering). Now all the system needs is for you to tell it who these people are. Just add one label per person<sup>3</sup> and it is able to name everyone in every photo, which is useful for searching photos.

Most semi-supervised learning algorithms are combinations of unsupervised and supervised algorithms. For example, a clustering algorithm may be used to group similar instances together, and then every unlabeled instance can be labeled with the most common label in its cluster. Once the whole dataset is labeled, it is possible to use any supervised learning algorithm.

### Self-supervised learning

Another approach to machine learning involves actually generating a fully labeled dataset from a fully unlabeled one. Again, once the whole dataset is labeled, any supervised learning algorithm can be used. This approach is called *self-supervised learning*.

For example, if you have a large dataset of unlabeled images, you can randomly mask a small part of each image and then train a model to recover the original image (Figure 1-12). During training, the masked images are used as the inputs to the model, and the original images are used as the labels.

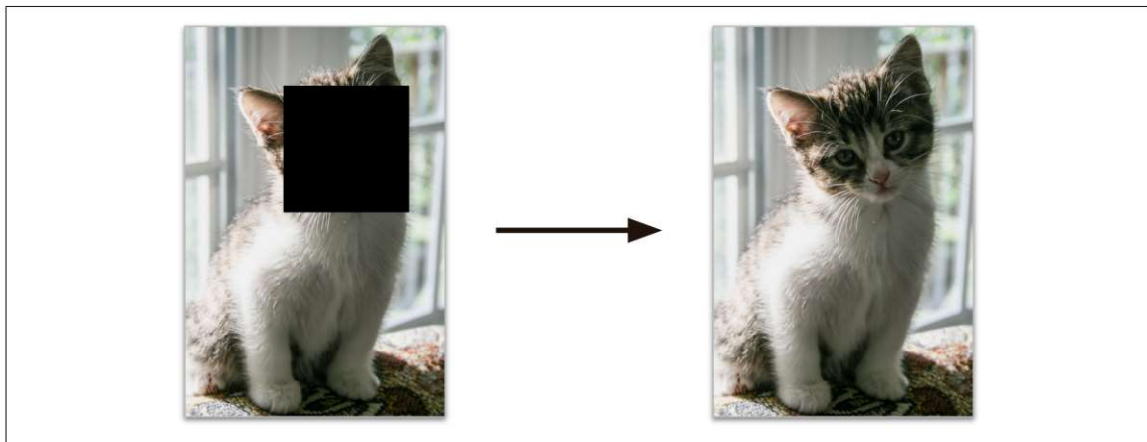


Figure 1-12. Self-supervised learning example: input (left) and target (right)

---

<sup>3</sup> That's when the system works perfectly. In practice it often creates a few clusters per person, and sometimes mixes up two people who look alike, so you may need to provide a few labels per person and manually clean up some clusters.