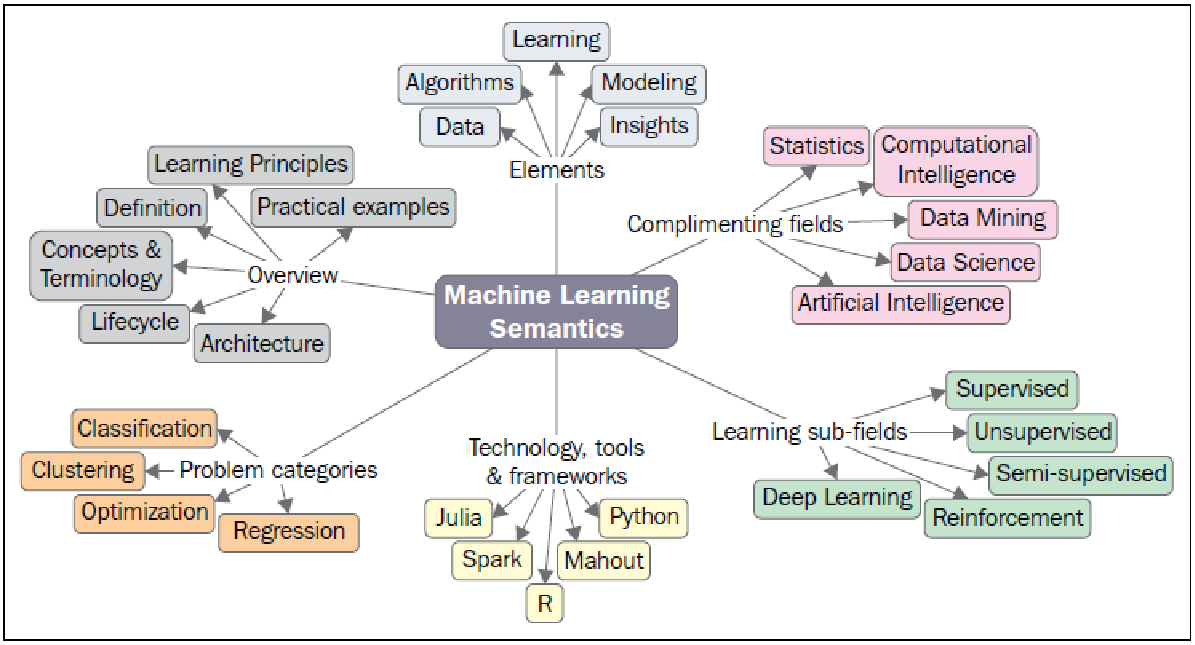
Introduction to Machine Learning



"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

– Tom M. Mitchell

"Machine learning is the training of a model from data that generalizes a decision against a performance measure."

– Jason Brownlee

"A branch of artificial intelligence in which a computer generates rules underlying or based on raw data that has been fed into it."

– Dictionary.com

"Machine learning is a scientific discipline that is concerned with the design and development of algorithms that allow computers to evolve behaviors based on empirical data, such as from sensor data or databases."

– Wikipedia

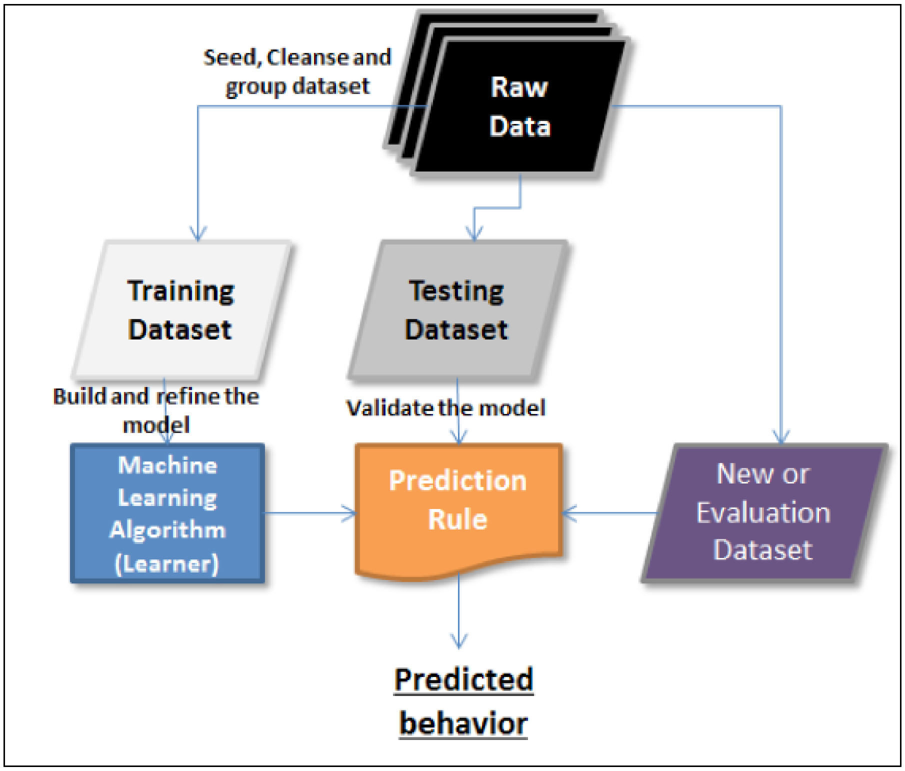
There are typically three phases for performing Machine learning:

Phase 1—Training Phase: This is the phase where training data is used to train the model by pairing the given input with the expected output. The output of this phase is the learning model itself.

Phase 2—Validation and Test Phase: This phase is to measure how good the learning model that has been trained is and estimate the model properties, such as error measures, recall, precision, and others. This phase uses a validation dataset, and the output is a sophisticated learning model.

Phase 3—Application Phase: In this phase, the model is subject to the real-world data for which the results need to be derived.

The following figure depicts how learning can be applied to predict the behavior:


Seed, Cleanse and group dataset
Raw Data
Training Dataset
Testing Dataset
Build and refine the model
Validate the model
Machine Learning Algorithm (Learner)
Prediction Rule
New or Evaluation Dataset
Predicted behavior

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:

- Whether or not they are trained with human supervision supervised, unsupervised, semisupervised, and Reinforcement Learning)

- 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 detect patterns in the training data and build 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-theart spam filter may learn on the fly using a deep neural network model trained using examples of spam and ham; this makes it an online, model-based, supervised learning system.

Supervised/Unsupervised Learning

Machine Learning systems can be classified according to the amount and type of supervision they get during training. There are four major categories: supervised learning, unsupervised learning, semisupervised learning, and Reinforcement Learning.

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

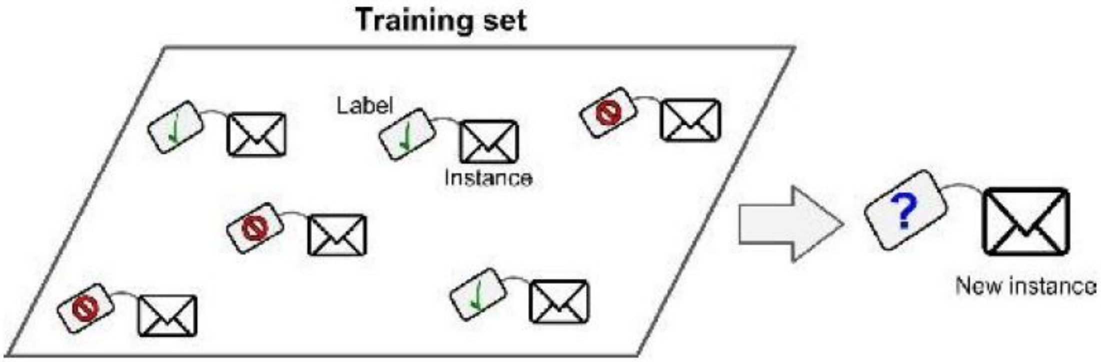


Figure 1-5. A labeled training set for supervised learning (e.g., spam classification)

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.) called predictors. This sort of task is called regression (Figure 1-6).1 To train the system, you need to give it many examples of cars, including both their predictors and their labels (i.e., their prices).

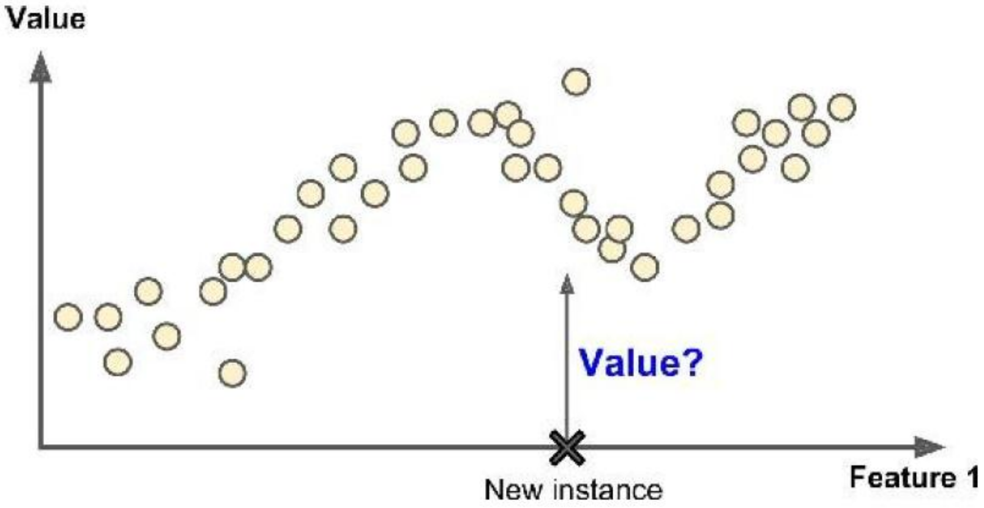


Figure 1-6. Regression

In **supervised learning**, the **dataset** is the collection of **labeled examples** {(**x***i, yi*)}*N* *i*=1. Each element **x***i* among *N* is called a **feature vector**. A feature vector is a vector in which each dimension *j* = 1*, . . . ,D* contains a value that describes the example somehow. That value is called a **feature** and is denoted as *x*(*j*). For instance, if each example **x** in our collection represents a person, then the first feature, *x*(1), could contain height in cm, the second feature, *x*(2), could contain weight in kg, *x*(3) could contain gender, and so on. For all examples in the dataset, the feature at position *j* in the feature vector always contains the same kind of information. It means that if *x* (2) *i* contains weight in kg in some example **x***i*, then *x* (2) *k* will also contain weight in kg in every example **x***k*, *k* = 1*, . . . ,N*. The **label** *yi* can be either an element belonging to a finite set of **classes** {1*,* 2*, . . . ,C*}, or a real number, or a more complex structure, like a vector, a matrix, a tree, or a graph. Unless otherwise stated, in this book *yi* is either one of a finite set of classes or a real number2. You can see a class as a category to which an example belongs. For instance, if your examples are email messages and your problem is spam detection, then you have two classes {*spam, not_spam*}.

The goal of a **supervised learning algorithm** is to use the dataset to produce a **model** that takes a feature vector **x** as input and outputs information that allows deducing the label for this feature vector. For instance, the model created using the dataset of people could take as input a feature vector describing a person and output a probability that the person has cancer.

Here are some of the most important **supervised learning algorithms**:

- k-Nearest Neighbors
- Linear Regression
- Logistic Regression
- Support Vector Machines (SVMs)
- Decision Trees and Random Forests
- Neural networks

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.

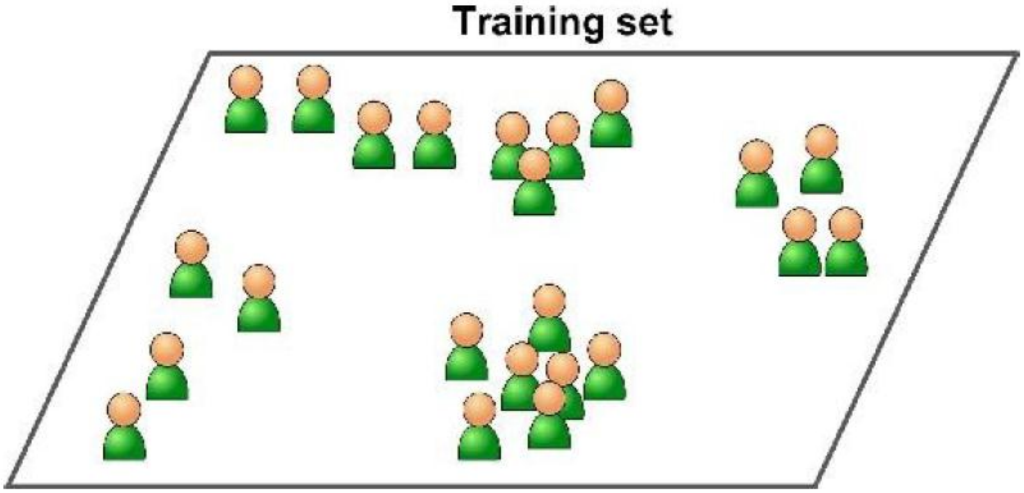


Figure 1-7. An unlabeled training set for unsupervised learning

In **unsupervised learning**, the dataset is a collection of **unlabeled examples** $\{\mathbf{x}i\}Ni=1$. Again, **x** is a feature vector, and the goal of an **unsupervised learning algorithm** is to create a **model** that takes a feature vector **x** as input and either transforms it into another vector or into a value that can be used to solve a practical problem. For example, in **clustering**, the model returns the id of the cluster for each feature vector in the dataset. In **dimensionality reduction**, the output of the model is a feature vector that has fewer features than the input **x**; in **outlier detection**, the output is a real number that indicates how **x** is different from a “typical” example in the dataset.

Examples of unsupervised learning algorithms:

- Clustering
- k-Means
- Hierarchical Cluster Analysis (HCA)
- Expectation Maximization

Visualization and dimensionality reduction

- Principal Component Analysis (PCA)
- Kernel PCA
- Locally-Linear Embedding (LLE)
- t-distributed Stochastic Neighbor Embedding (t-SNE)

Association rule learning

- Apriori
- Eclat

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 males who love comic books and generally read your blog in the evening, while 20% are young sci-fi lovers who visit during the weekends, and so on. 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.

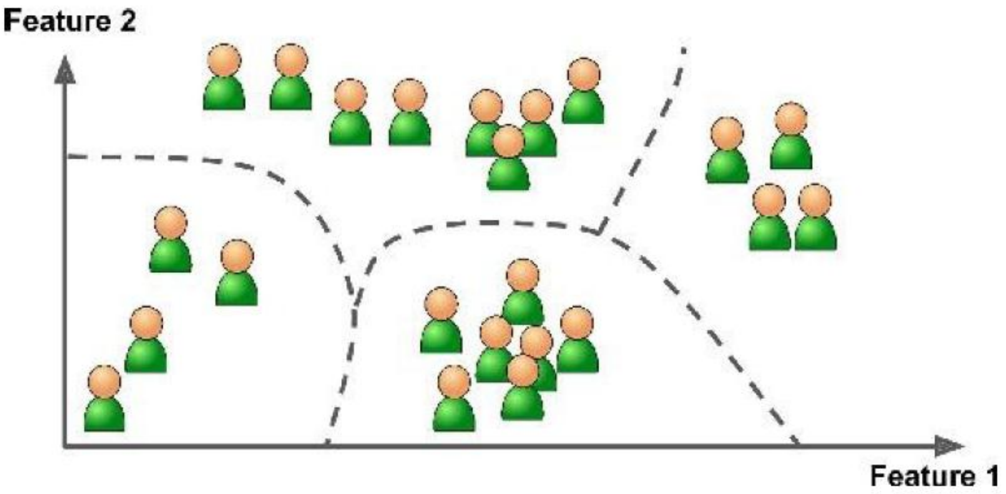


Figure 1-8. Clustering

Visualization algorithms are also good examples of unsupervised learning algorithms: 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 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 very 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.

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 trained with normal instances, and 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).

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 each other.

Semisupervised learning

Some algorithms can deal with partially labeled training data, usually a lot of unlabeled data and a little bit of labeled data. This is called semisupervised learning (Figure 1-11).

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 one label per person,4 and it is able to name everyone in every photo, which is useful for searching photos.

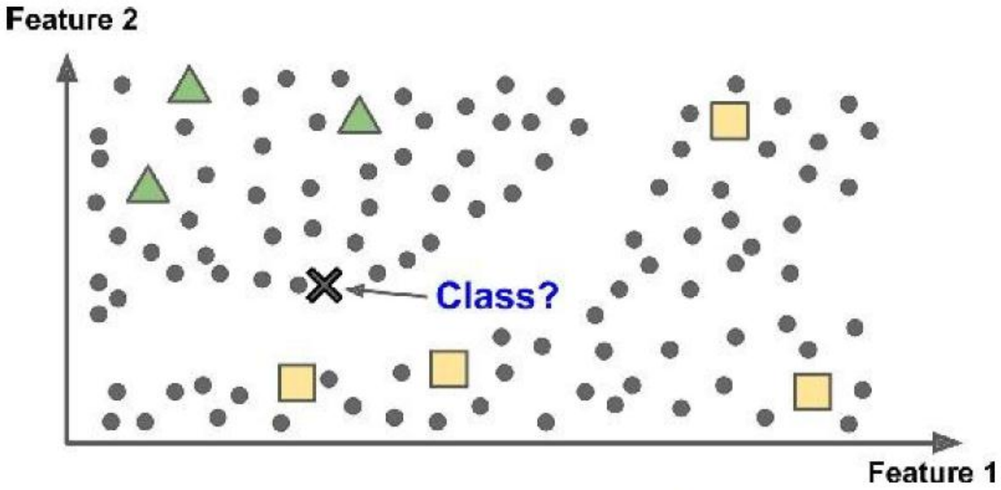


Figure 1-11. Semisupervised learning

Most semisupervised learning algorithms are combinations of unsupervised and supervised algorithms. For example, deep belief networks (DBNs) are based on unsupervised components called restricted Boltzmann machines (RBMs) stacked on top of one another. RBMs are trained sequentially in an unsupervised manner, and then the whole system is fine-tuned using supervised learning techniques.

Reinforcement Learning

Reinforcement Learning is a very different beast. The learning system, called an agent in this context, can observe the environment, select and perform actions, and get rewards in return (or penalties in the form of negative rewards, as in Figure 1-12). It must then learn by itself what is the best strategy, called a policy, to get the most reward over time. A policy defines what action the agent should choose when it is in a given situation.

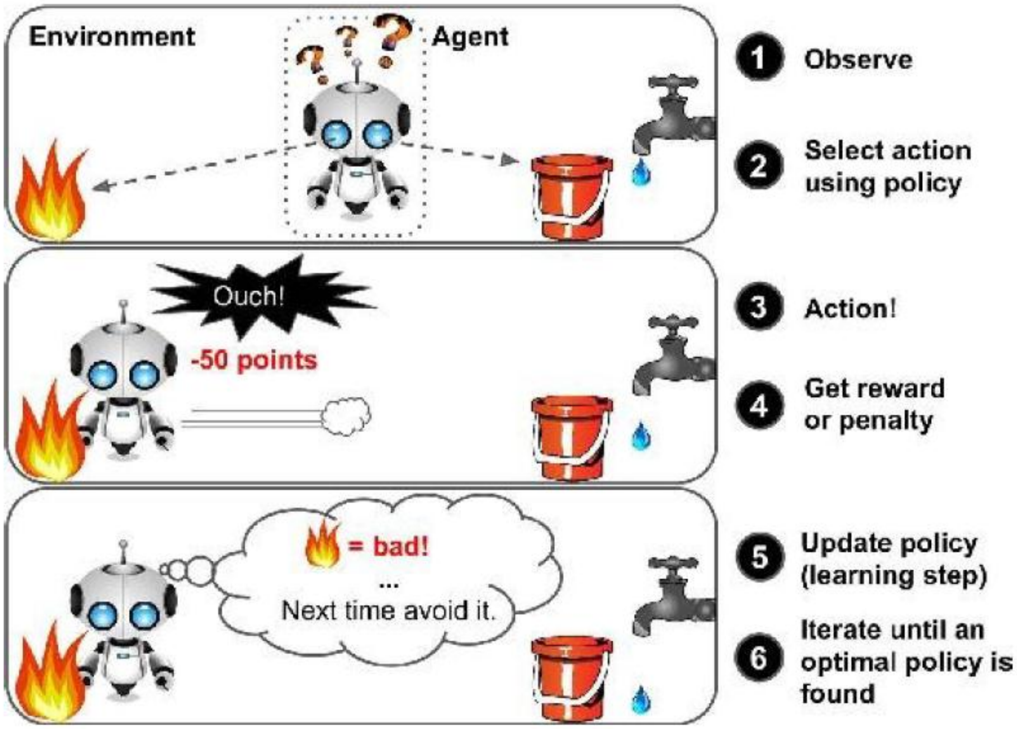

Environment
Agent
Ouch!
-50 points
= bad!
...
Next time avoid it.
1 Observe
2 Select action using policy
3 Action!
4 Get reward or penalty
5 Update policy (learning step)
6 Iterate until an optimal policy is found

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).

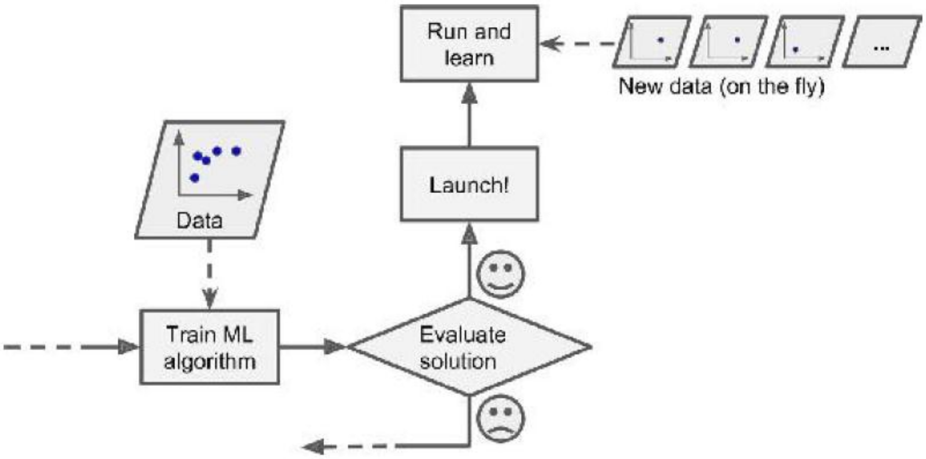


Figure 1-13. Online learning

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).

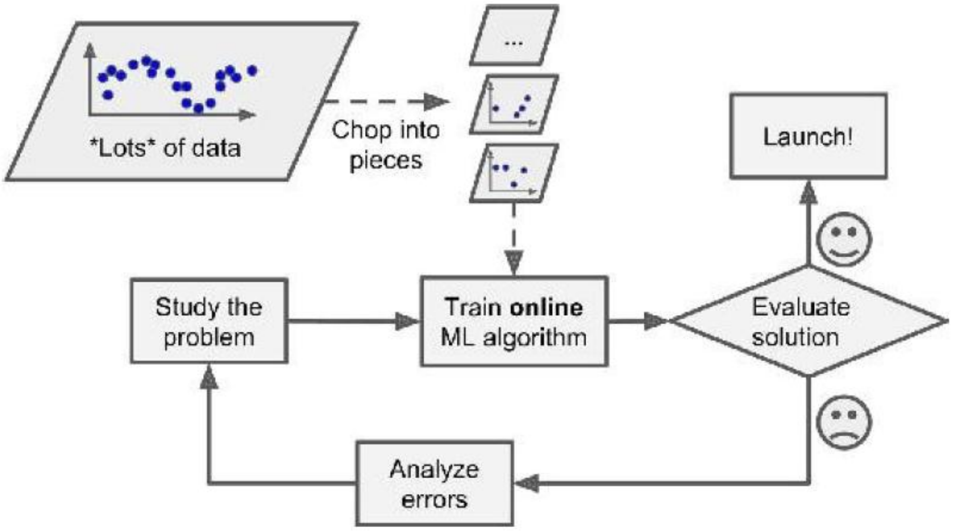


Figure 1-14. Using online learning to handle huge datasets

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 earning.

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).

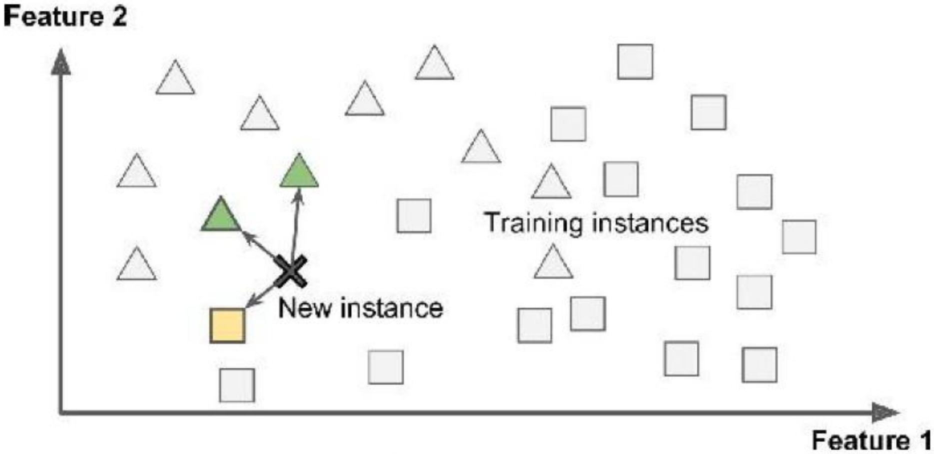


Figure 1-15. Instance-based learning

Model-based learning

Another way to generalize from a set of examples is to build a model of these examples, then use that model to make predictions. This is called model-based learning (Figure 1-16).

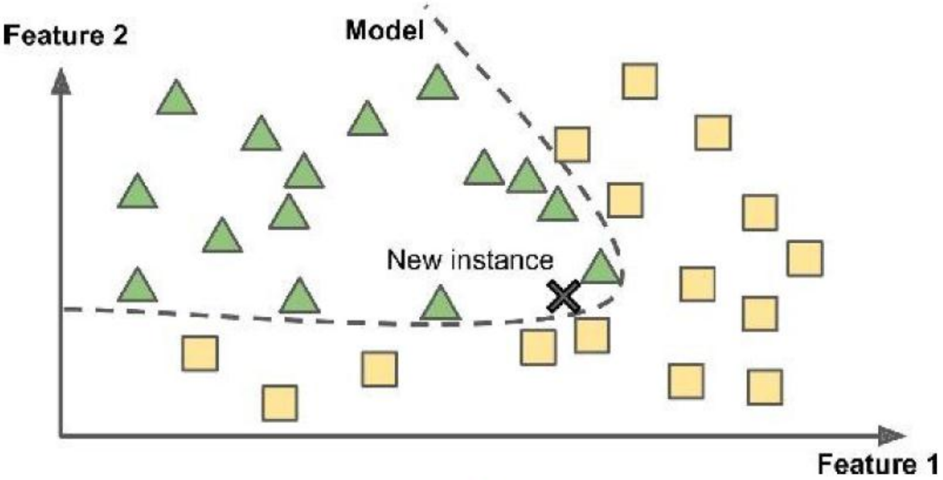


Figure 1-16. Model-based learning

3 Fundamental Algorithms

In this chapter, I describe five algorithms which are not just the most known but also either very effective on their own or are used as building blocks for the most effective learning algorithms out there.

3.1 Linear Regression

Linear regression is a popular regression learning algorithm that learns a model which is a linear combination of features of the input example.

3.2 Logistic Regression

The first thing to say is that logistic regression is not a regression, but a classification learning algorithm. The name comes from statistics and is due to the fact that the mathematical formulation of logistic regression is similar to that of linear regression. I explain logistic regression on the case of binary classification. However, it can naturally be extended to multiclass classification.

3.3 Decision Tree Learning

A decision tree is an acyclic **graph** that can be used to make decisions. In each branching node of the graph, a specific feature *j* of the feature vector is examined. If the value of the feature is below a specific threshold, then the left branch is followed; otherwise, the right branch is followed. As the leaf node is reached, the decision is made about the class to which the example belongs. As the title of the section suggests, a decision tree can be learned from data.

3.4 Support Vector Machine

I already presented SVM in the introduction, so this section only fills a couple of blanks. Two critical questions need to be answered:

1. What if there’s noise in the data and no hyperplane can perfectly separate positive examples from negative ones?

2. What if the data cannot be separated using a plane, but could be separated by a higher-order polynomial?

You can see both situations depicted in Figure 5. In the left case, the data could be separated by a straight line if not for the noise (outliers or examples with wrong labels). In the right case, the decision boundary is a circle and not a straight line.

3.5 k-Nearest Neighbors

k-Nearest Neighbors (kNN) is a non-parametric learning algorithm. Contrary to other learning algorithms that allow discarding the training data after the model is built, kNN keeps all training examples in memory. Once a new, previously unseen example **x** comes in, the kNN algorithm finds *k* training examples closest to **x** and returns the majority label, in case of classification, or the average label, in case of regression.

The closeness of two examples is given by a distance function. For example, Euclidean distance seen above is frequently used in practice. Another popular choice of the distance function is the negative **cosine similarity**.

Main Challenges of Machine Learning

In short, since your main task is to select a learning algorithm and train it on some data, the two things that can go wrong are “bad algorithm” and “bad data.” Let’s start with examples of bad data.

Insufficient Quantity of Training Data

For a toddler to learn what an apple is, all it takes is for you to point to an apple and say “apple” (possibly repeating this procedure a few times). Now the child is able to recognize apples in all sorts of colors and shapes. Genius. Machine Learning is not quite there yet; it takes a lot of data for most Machine Learning algorithms to work properly. Even for very simple problems you typically need thousands of examples, and for complex problems such as image or speech recognition you may need millions of examples (unless you can reuse parts of an existing model).

Nonrepresentative Training Data

In order to generalize well, it is crucial that your training data be representative of the new cases you want to generalize to. This is true whether you use instance-based learning or model-based learning. For example, the set of countries we used earlier for training the linear model was not perfectly representative; a few countries were missing. Figure 1-21 shows what the data looks like when you add the missing countries.

If you train a linear model on this data, you get the solid line, while the old model is represented by the dotted line. As you can see, not only does adding a few missing countries significantly alter the model, but it makes it clear that such a simple linear model is probably never going to work well. It seems that very rich countries are not happier than moderately rich countries (in fact they seem unhappier), and conversely some poor countries seem happier than many rich countries. By using a nonrepresentative training set, we trained a model that is unlikely to make accurate predictions, especially for very poor and very rich countries. It is crucial to use a training set that is representative of the cases you want to generalize to. This is often harder than it sounds: if the sample is too small, you will have sampling noise (i.e., nonrepresentative data as a result of chance), but even very large samples can be nonrepresentative if the sampling method is flawed. This is called sampling bias.

Poor-Quality Data

Obviously, if your training data is full of errors, outliers, and noise (e.g., due to poor-quality measurements), it will make it harder for the system to detect the underlying patterns, so your system is less likely to perform well. It is often well worth the effort to spend time cleaning up your training data. The truth is, most data scientists spend a significant part of their time doing just that. For example: If some instances are clearly outliers, it may help to simply discard them or try to fix the errors manually. If some instances are missing a few features (e.g., 5% of your customers did not specify their age),you must decide whether you want to ignore this attribute altogether, ignore these instances, fill in the missing values (e.g., with the median age), or train one model with the feature and one model without it, and so on.

Irrelevant Features

As the saying goes: garbage in, garbage out. Your system will only be capable of learning if the training data contains enough relevant features and not too many irrelevant ones. A critical part of the success of a Machine Learning project is coming up with a good set of features to train on. This process, called feature engineering, involves:

Feature selection: selecting the most useful features to train on among existing features.

Feature extraction: combining existing features to produce a more useful one (as we saw earlier,dimensionality reduction algorithms can help).Creating new features by gathering new data. Now that we have looked at many examples of bad data,let's look at a couple of examples of bad algorithms.

Overfitting the Training Data

Say you are visiting a foreign country and the taxi driver rips you off. You might be tempted to say that all taxi drivers in that country are thieves. Overgeneralizing is something that we humans do all too often, and unfortunately machines can fall into the same trap if we are not careful. In Machine Learning this is called overfitting: it means that the model performs well on the training data, but it does not generalize well. Figure 1-22 shows an example of a high-degree polynomial life satisfaction model that strongly overfits the training data. Even though it performs much better on the training data than the simple linear model, would you really trust its predictions?

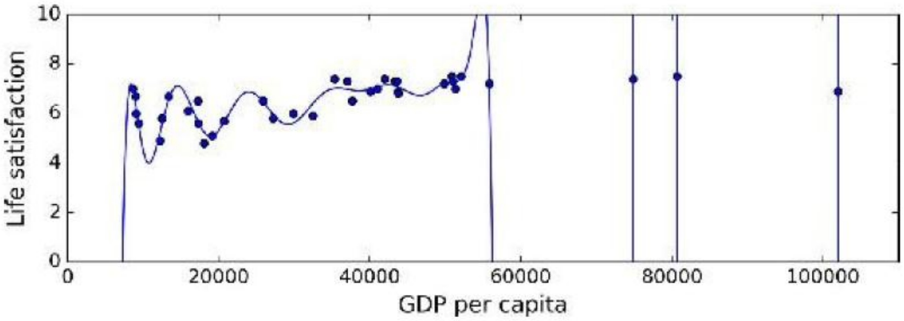


Figure 1-22. Overfitting the training data

Complex models such as deep neural networks can detect subtle patterns in the data, but if the training set is noisy, or if it is too small (which introduces sampling noise), then the model is likely to detect patterns in the noise itself. Obviously these patterns will not generalize to new instances. For example, say you feed your life satisfaction model many more attributes, including uninformative ones such as the country's name. In that case, a complex model may detect patterns like the fact that all countries in the training data with a w in their name have a life satisfaction greater than 7: New Zealand (7.3), Norway (7.4), Sweden (7.2), and Switzerland (7.5). How confident are you that the W-satisfaction rule generalizes to Rwanda or Zimbabwe? Obviously this pattern occurred in the training data by pure chance, but the model has no way to tell whether a pattern is real or simply the result of noise in the data.

Constraining a model to make it simpler and reduce the risk of overfitting is called regularization. For example, the linear model we defined earlier has two parameters, θ 0 and θ 1. This gives the learning algorithm two degrees of freedom to adapt the model to the training data: it can tweak both the height (θ 0) and the slope (θ 1) of the line. If we forced θ 1 = 0, the algorithm would have only one degree of freedom and would have a much harder time fitting the data properly: all it could do is move the line up or down to get as close as possible to the training instances, so it would end up around the mean. A very simple model indeed! If we allow the algorithm to modify θ 1 but we force it to keep it small, then the learning algorithm will effectively have somewhere in between one and two degrees of freedom. It will produce a simpler model than with two degrees of freedom, but more complex than with just one. You want to find the right balance between fitting the data perfectly and keeping the model simple enough to ensure that it will generalize well.Figure 1-23 shows three models: the dotted line represents the original model that was trained with a few countries missing, the dashed line is our second model trained with all countries, and the solid line is a linear model trained with the same data as the first model but with a regularization constraint. You can see that regularization forced the model to have a smaller slope, which fits a bit less the training data that the model was trained on, but actually allows it to generalize better to new examples.

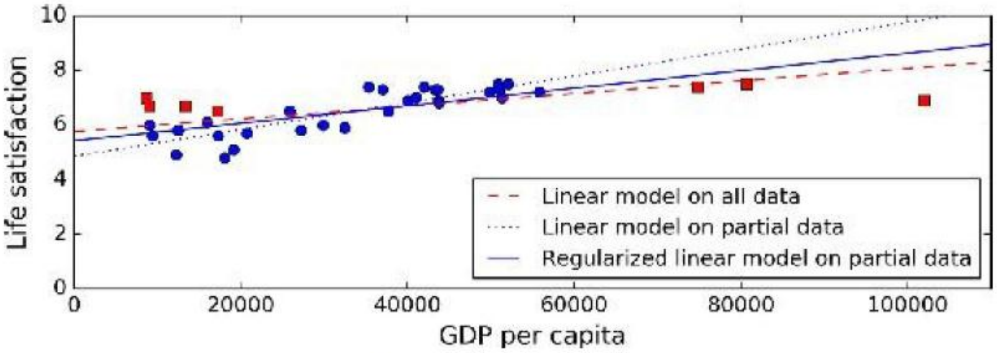


Figure 1-23. Regularization reduces the risk of overfitting

The amount of regularization to apply during learning can be controlled by a hyperparameter. A hyperparameter is a parameter of a learning algorithm (not of the model). As such, it is not affected by the learning algorithm itself; it must be set prior to training and remains constant during training. If you set the regularization hyperparameter to a very large value, you will get an almost flat model (a slope close to zero); the learning algorithm will almost certainly not overfit the training data, but it will be less likely to find a good solution. Tuning hyperparameters is an important part of building a Machine Learning system

Underfitting the Training Data

As you might guess, underfitting is the opposite of overfitting: it occurs when your model is too simple to learn the underlying structure of the data. For example, a linear model of life satisfaction is prone to underfit; reality is just more complex than the model, so its predictions are bound to be inaccurate, even on the training examples. The main options to fix this problem are:

- Selecting a more powerful model, with more parameters Feeding better features to the learning algorithm (feature engineering)

- Reducing the constraints on the model (e.g., reducing the regularization hyperparameter)