Chapter 1 - The Machine Learning Landscape

August 4, 2021

1 What Is Machine Learning?

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

A 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

---

2 Why Use Machine Learning?

Machine Learning is great for:

- Problems for which existing solutions require a lot of fine-tuning or long lists of rules: one Machine Learning algorithm 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 adapt to new data.
- Getting insights about complex problems and large amounts of data.

---

3 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:

- 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 by detecting patterns in the training data and building a predictive model, much like scientists do (instance-based versus model-based learning)

3.1 Supervised/Unsupervised Learning

Machine Learning systems can be classified according to the amount and type of supervision they get during training.

3.1.1 Supervised learning

In supervised learning, the training set you feed to the algorithm includes the desired solutions, called labels.

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 1-1).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).

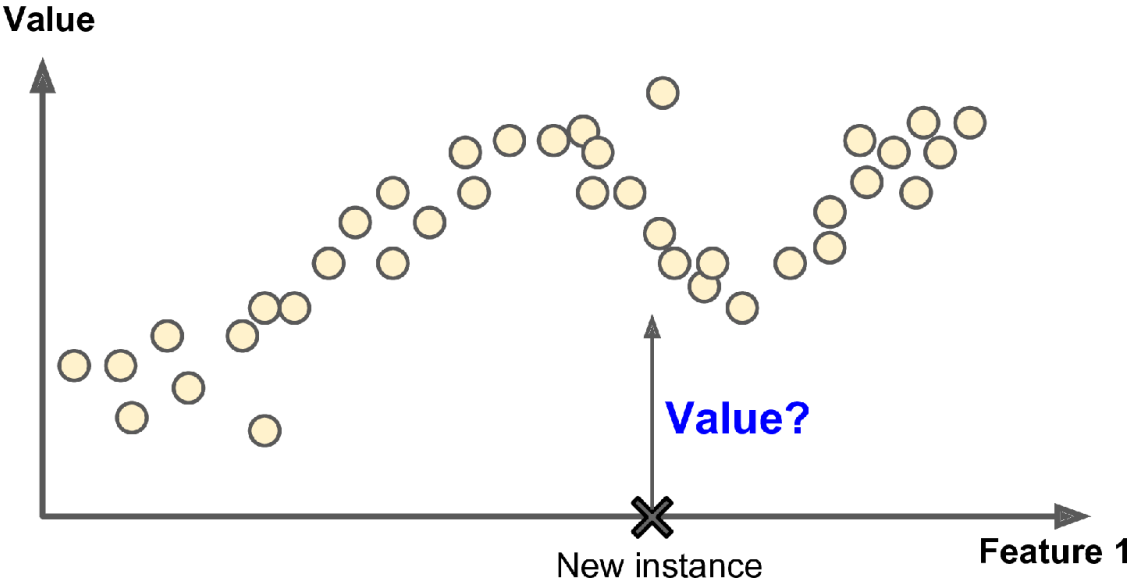

Figure 1-1 . A regression problem: predict a value, given an input feature (there are usually multiple input features, and sometimes multiple output values)

In Machine Learning, an attribute is a data type (e.g., “mileage”), while a feature has several meanings, depending on the context, but generally means an attribute plus its value (e.g., “mileage = 15,000”). Many people use the words attribute and feature interchangeably

3.1.2 Unsupervised learning

In unsupervised learning, as you might guess, the training data is unlabeled. The system tries to learn without a teacher. Here are some of the most important unsupervised learning algorithms:

- Clustering
- Anomaly detection and novelty detection
- Visualization and dimensionality reduction
- Association rule learning

Visualization algorithms are 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 that you can understand how the data is organized and perhaps identify unsuspected patterns.

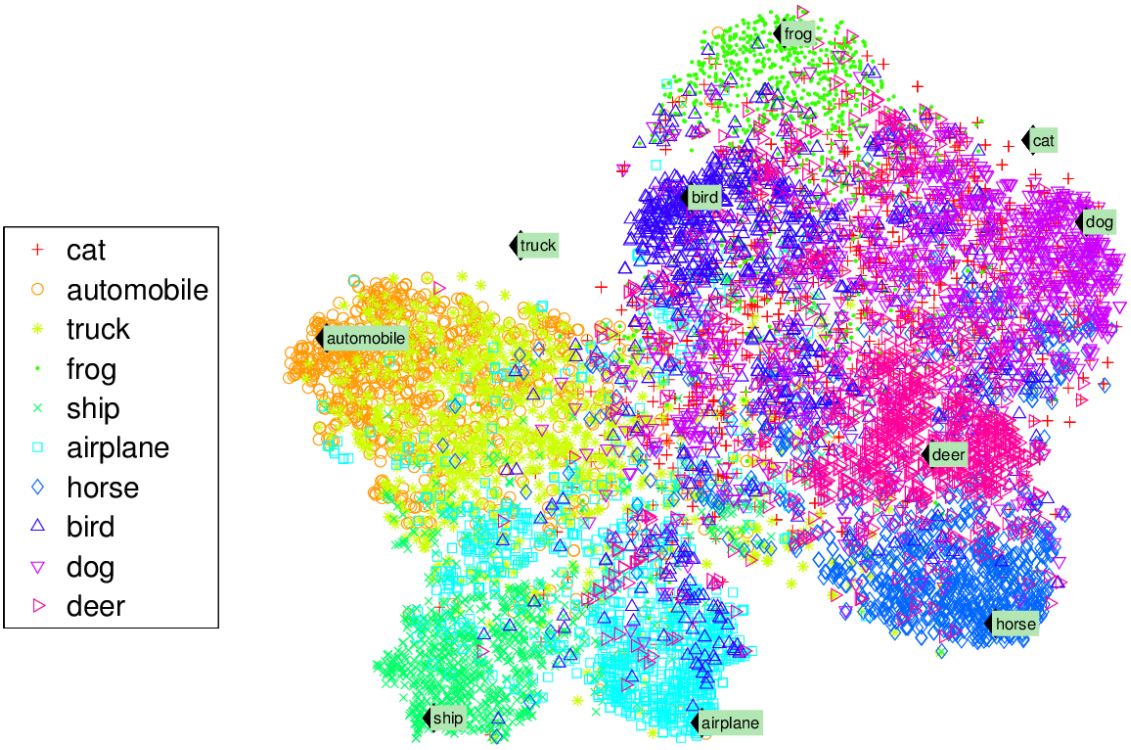

Figure 1-2. Example of a t-SNE visualization highlighting semantic clusters

Notice how animals are rather well separated from vehicles and how horses are close to deer but far from birds.

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. It is often a good idea to try to reduce the dimension of 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.

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.

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.

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.

3.1.3 Semisupervised 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 semisupervised learning. Google Photos, are good examples of this. Once you upload all your family photos to the service, it automatically creates a few clusters per person (the unsupervised part). Now all the system needs is for you to tell it who these people are. Just add one label per person (That's when the system works perfectly. In practice it 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) and it is able to name everyone in every photo, which is useful for searching photos.

Most semisupervised learning algorithms are combinations of unsupervised and supervised algorithms.

3.1.4 Reinforcement Learning

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 shown in Figure 1-3). 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.

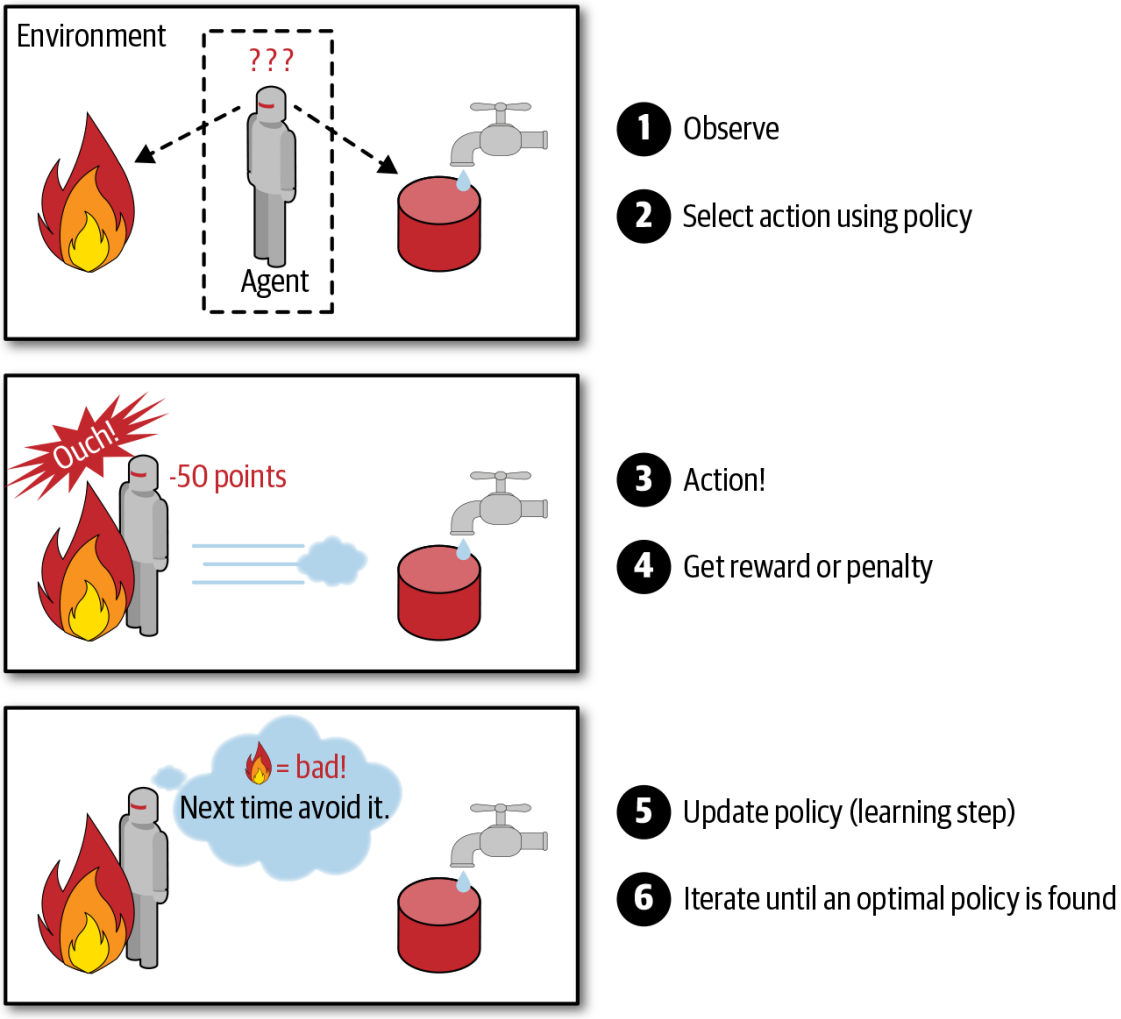

Figure 1-3. Reinforcement Learning

3.2 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.

3.2.1 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, 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, so even a batch learning system can adapt to change. 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, 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, 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.

3.2.2 Online learning

In online learning, you train the system incrementally by feeding it data instances sequentially, either individually or in 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.

Online learning is great for systems that receive data as a continuous flow 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).

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.

Out-of-core learning 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.

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 and what it learned from them. 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 (outliers).

A big challenge with online learning is that if bad data is fed to the system, the system's performance will gradually decline. If it's a live system, your clients will notice. 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).

3.3 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 make good predictions for (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.

3.3.1 Instance-based learning

This is called instance-based learning: the system learns the examples by heart, then generalizes to new cases by using a similarity measure to compare them to the learned examples (or a subset of them).

3.3.2 Model-based learning

Another way to generalize from a set of examples is to build a model of these examples and then use that model to make predictions. This is called model-based learning.

Before you can use your model, you need to know which parameter values will make your model perform best? To answer this question, you need to specify a performance measure. You can either define a utility function (or fitness function) that measures how good your model is, or you can define a cost function that measures how bad it is. For Linear Regression problems, people typically use a cost function that measures the distance between the linear model's predictions and the training examples; the objective is to minimize this distance.

Confusingly, the same word "model" can refer to a type of model (e.g., Linear Regression), to a fully specified model architecture (e.g., Linear Regression with one input and one output), or to the final trained model ready to be used for predictions (e.g., Linear Regression with one input and one output, using $0 = 4.85$ and $1 = 4.91 \times 10–5$). Model selection consists in choosing the type of model and fully specifying its architecture. Training a model means running an algorithm to find the model parameters that will make it best fit the training data.

In summary, a typical Machine Learning project looks like this:

- You studied the data.
- You selected a model.
- You trained it on the training data (i.e., the learning algorithm searched for the model parameter values that minimize a cost function).
- Finally, you applied the model to make predictions on new cases (this is called inference), hoping that this model will generalize well.

3.4 Main Challenges of Machine Learning

The two things that can go wrong are "bad algorithm" and "bad data."

3.4.1 Examples of bad data.

Insufficient Quantity of Training Data Machine Learning 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).

The Unreasonable Effectiveness of Data: In a famous paper published in 2001, Microsoft researchers Michele Banko and Eric Brill showed that very different Machine Learning algorithms, including fairly simple ones, performed almost identically well on a complex problem of natural language disambiguation (For example, knowing whether to write "to," "two," or "too," depending on the context) once they were given enough data (as you can see in Figure 1-4).

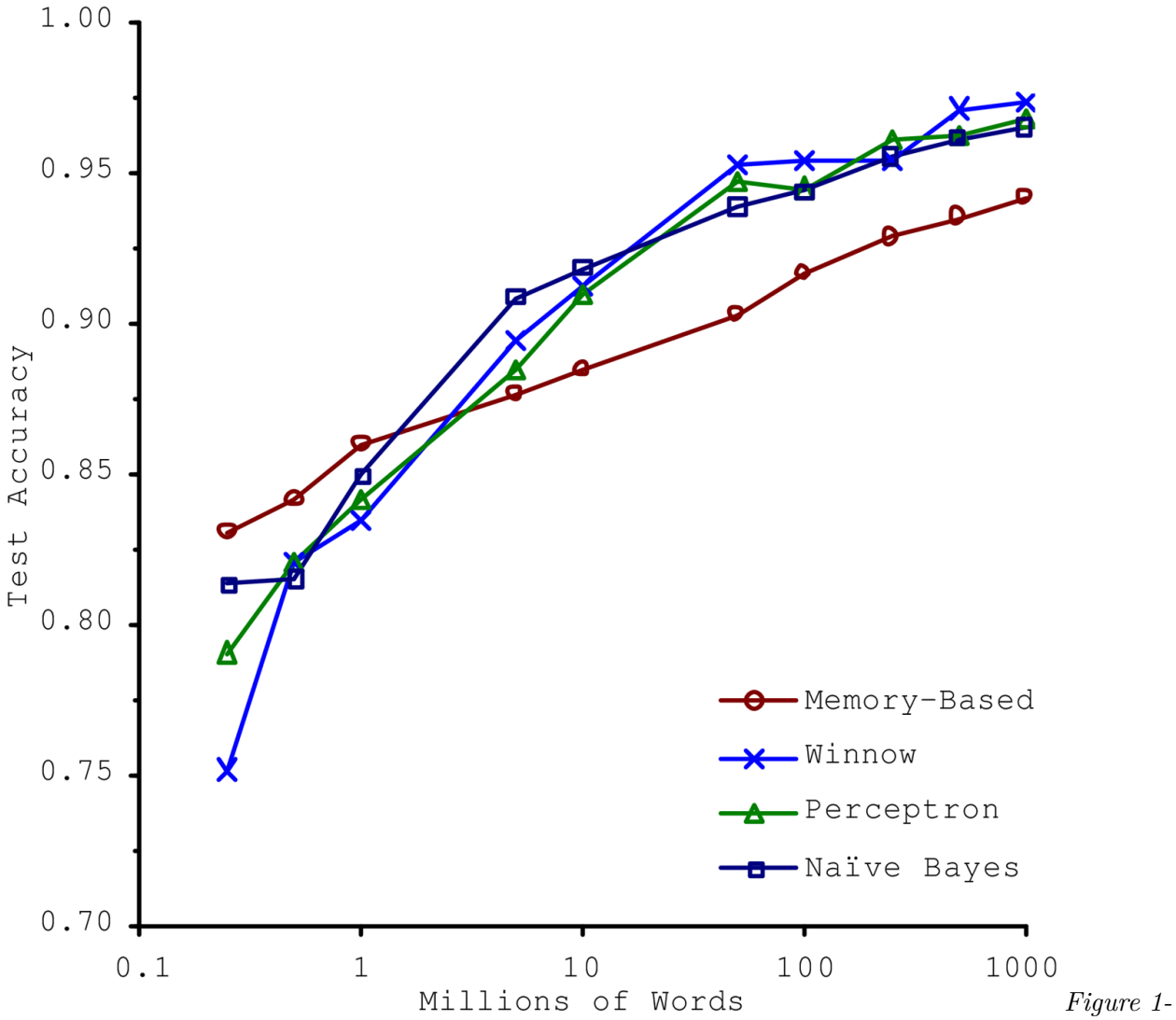

Figure 1-4. The importance of data versus algorithms

As the authors put it, "these results suggest that we may want to reconsider the tradeoff between spending time and money on algorithm development versus spending it on corpus development."

The idea that data matters more than algorithms for complex problems. It should be noted, however, that small- and medium-sized datasets are still very common, and it is not always easy or cheap to get extra training data—so don't abandon algorithms just yet.

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.

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. The following are a couple of examples of when you'd want to clean up training data:

- 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.

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 the following steps:

- 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

3.4.2 Examples of Bad Algorithms

Overfitting the Training Data It means that the model performs well on the training data, but it does not generalize well. Overfitting happens when the model is too complex relative to the amount and noisiness of the training data. Here are possible solutions:

- Simplify the model by selecting one with fewer parameters (e.g., a linear model rather than a high-degree polynomial model), by reducing the number of attributes in the training data, or by constraining the model.
- Gather more training data.
- Reduce the noise in the training data (e.g., fix data errors and remove outliers).

Constraining a model to make it simpler and reduce the risk of overfitting is called regularization.

You want to find the right balance between fitting the training data perfectly and keeping the model simple enough to ensure that it will generalize well.

The amount of regularization to apply during learning can be controlled by a hyper-parameter. 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, 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. Here are the main options for fixing this problem:

- Select a more powerful model, with more parameters.
- Feed better features to the learning algorithm (feature engineering).
- Reduce the constraints on the model (e.g., reduce the regularization hyperparameter).

3.5 Stepping Back

- Machine Learning is about making machines get better at some task by learning from data, instead of having to explicitly code rules.
- There are many different types of ML systems: supervised or not, batch or online, instance-based or model-based.
- In an ML project you gather data in a training set, and you feed the training set to a learning algorithm. If the algorithm is model-based, it tunes some parameters to fit the model to the training set (i.e., to make good predictions on the training set itself), and then hopefully it will be able to make good predictions on new cases as well. If the algorithm is instance-based, it just learns the examples by heart and generalizes to new instances by using a similarity measure to compare them to the learned instances.
- The system will not perform well if your training set is too small, or if the data is not representative, is noisy, or is polluted with irrelevant features (garbage in, garbage out). Lastly, your model needs to be neither too simple (in which case it will underfit) nor too complex (in which case it will overfit).

3.6 Testing and Validating

The only way to know how well a model will generalize to new cases is to actually try it out on new cases. A good option is to split your data into two sets: the training set and the test set. As these names imply, you train your model using the training set, and you test it using the test set. The error rate on new cases is called the generalization error (or out-of-sample error), and by evaluating your model on the test set, you get an estimate of this error. This value tells you how well your model will perform on instances it has never seen before.

If the training error is low (i.e., your model makes few mistakes on the training set) but the generalization error is high, it means that your model is overfitting the training data.

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.

3.6.1 Hyperparameter Tuning and Model Selection

How can you decide between two types of models (say, a linear model and a polynomial model)? One option is to train both and compare how well they generalize using the test set.

How do you choose the value of the regularization hyperparameter? A common solution to this problem is called holdout validation: you simply hold out part of the training set to evaluate several candidate models and select the best one. The new held-out set is called the validation set (or sometimes the development set, or dev set). More specifically, you train multiple models with various hyperparameters on the reduced training set (i.e., the full training set minus the validation set), and you select the model that performs best on the validation set. After this holdout validation process, you train the best model on the full training set (including the validation set), and this gives you the final model. Lastly, you evaluate this final model on the test set to get an estimate of the generalization error.

This solution usually works quite well. However, if the validation set is too small, then model evaluations will be imprecise: you may end up selecting a suboptimal model by mistake. Conversely, if the validation set is too large, then the remaining training set will be much smaller than the full training set. Why is this bad? It would be like selecting the fastest sprinter to participate in a marathon. One way to solve this problem is to perform repeated cross-validation, using many small validation sets. Each model is evaluated once per validation set after it is trained on the rest of the data. By averaging out all the evaluations of a model, you get a much more accurate measure of its performance. There is a drawback, however: the training time is multiplied by the number of validation sets.

3.6.2 Data Mismatch

In some cases, it's easy to get a large amount of data for training, but this data probably won't be perfectly representative of the data that will be used in production (images from a camera app vs. images from the web). In this case, the most important rule to remember is that the validation set and the test set must be as representative as possible of the data you expect to use in production, so they should be composed exclusively of representative pictures: you can shuffle them and put half in the validation set and half in the test set (making sure that no duplicates or near-duplicates end up in both sets).

After training your model on the web pictures, if you observe that the performance of the model on the validation set is disappointing, you will not know whether this is because your model has overfit the training set, or whether this is just due to the mismatch between the web pictures and the mobile app pictures. One solution is to hold out some of the training pictures (from the web) in yet another set called the train-dev set. 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 it performs well, then the model is not overfitting the training set. If it performs poorly on the validation set, 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.

3.6.3 No Free Lunch Theorem

A model is a simplified version of the observations. The simplifications are meant to discard the superfluous details that are unlikely to generalize to new instances. To decide what data to discard

and what data to keep, you must make assumptions. For example, a linear model makes the assumption 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, It was 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. The only reasonable way to know for sure which model is best is to 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.