* **Machine Learning is great for:**

**1**. Problems for which existing solutions require a lot of hand-tuning or long lists of rules: one Machine Learning algorithm can often simplify code and perform better.

 **2**. Complex problems for which there is no good solution at all using a traditional approach: the best Machine Learning techniques can find a solution.

**3**. Fluctuating environments: a Machine Learning system can adapt to new data.

**4**. Getting insights about complex problems and large amounts of data.

* 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.**

* 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**

[ Some neural network architectures can be unsupervised, such as autoencoders and restricted Boltzmann machines. They can also be semisupervised, such as in deep belief networks and unsupervised pretraining. ]

* some of the most important 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**

**Supervised learning:**

 In supervised learning, the training data 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.

**Unsupervised Learning:**

 Here, the training data is unlabelled.

 Visualization algorithms are good examples of unsupervised learning algorithms.

 **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.

 Another important unsupervised task is – **anomaly detection**, removing unused or extra data.

 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.

**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.

 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, and it is able to name everyone in every photo, which is useful for searching photos.

**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).

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.

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.

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

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.

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.

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.

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.

**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.**

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.

**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.**

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.

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**.

**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.

**Overfitting** happens when the model is too complex relative to the amount and noisiness of the training data. The possible solutions are:

• To 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

• To gather more training data

• To 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**.

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 hyper parameter 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** 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 hyper‐ parameter)

**Testing and Validation :**

The only way to know how well a model will generalize to new cases is to actually try it out on new cases. One way to do that is to put your model in production and monitor how well it performs.

A better 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 estimation 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.**

So evaluating a model is simple enough: just use a test set. Now suppose you are hesitating between two models (say a linear model and a polynomial model): how can you decide? One option is to train both and compare how well they generalize using the test set.

Now suppose that the linear model generalizes better, but you want to apply some regularization to avoid overfitting. The question is: how do you choose the value of the regularization hyperparameter? One option is to train 100 different models using 100 different values for this hyperparameter. Suppose you find the best hyperparameter value that produces a model with the lowest generalization error, say just 5% error.

So you launch this model into production, but unfortunately it does not perform as well as expected and produces 15% errors. What just happened?

The problem is that you measured the generalization error multiple times on the test set, and you adapted the model and hyperparameters to produce the best model for that set. This means that the model is unlikely to perform as well on new data.

 A common solution to this problem is to have a second holdout set called the **validation set**. You train multiple models with various hyperparameters using the training set, you select the model and hyperparameters that perform best on the validation set, and when you’re happy with your model you run a single final test against the test set to get an estimate of the generalization error.

To avoid “wasting” too much training data in validation sets, a common technique is to use cross-validation: the training set is split into complementary subsets, and each model is trained against a different combination of these subsets and validated against the remaining parts. Once the model type and hyperparameters have been selected, a final model is trained using these hyperparameters on the full training set, and the generalized error is measured on the test set.