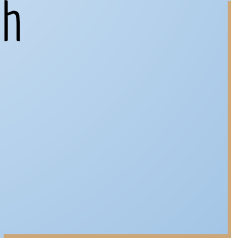# The Machine learning

Salar Mokhtari Laleh

Spring 2021

01_The machine learning landscape

01_The machine learning landscape

- <u>What's machine learning ?</u>
- <u>Why use machine learning ?</u>
- Types of machine learning systems
- Main challenges of machine learning
- Testing and validation
- Exercises

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

For example, your spam filter is a Machine Learning program that can learn to flag spam given examples of spam emails (e.g., flagged by users) and examples of regular (nonspam, also called "ham") emails. The examples that the system uses to learn are called the training set. Each training example is called a training instance (or sample). 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 Wikipedia, your computer has a lot more data, but it is not suddenly better at any task. Thus, downloading a copy of Wikipedia is not ML
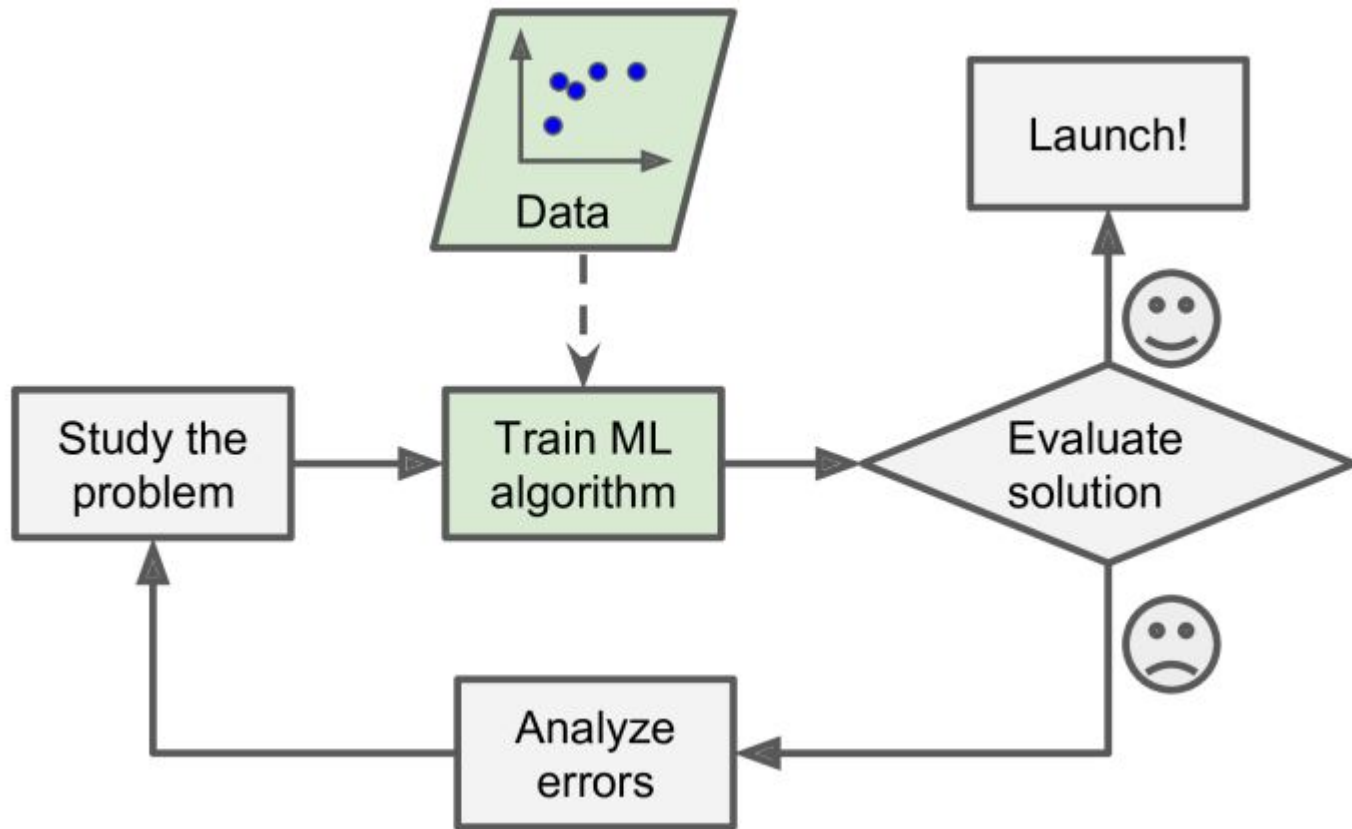
Why Use Machine Learning?

Consider how you would write a spam filter using traditional programming techniques
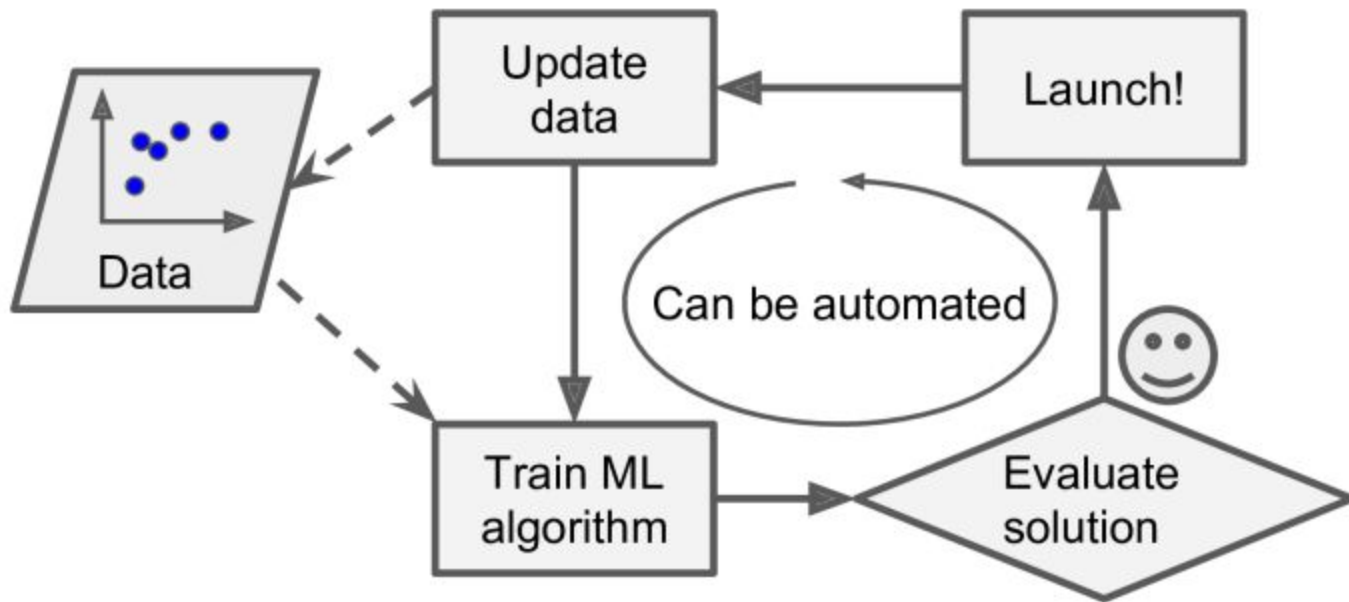
1. First you would look at what spam typically looks like. You might notice that some words or phrases (such as "4U," "credit card," "free," and "amazing") tend to come up a lot in the subject. Perhaps you would also notice a few other patterns in the sender's name, the email's body, and so on

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 are detected

3. You would test your program, and repeat steps 1 and 2 until it is good enough
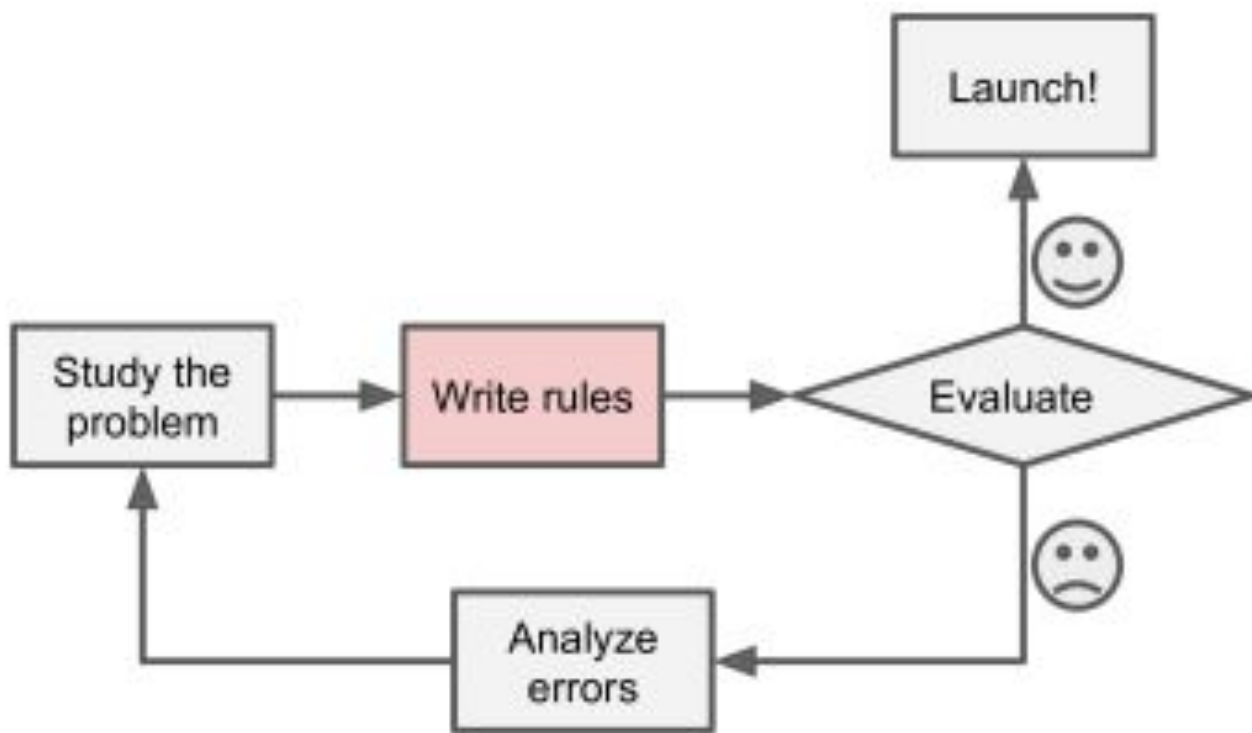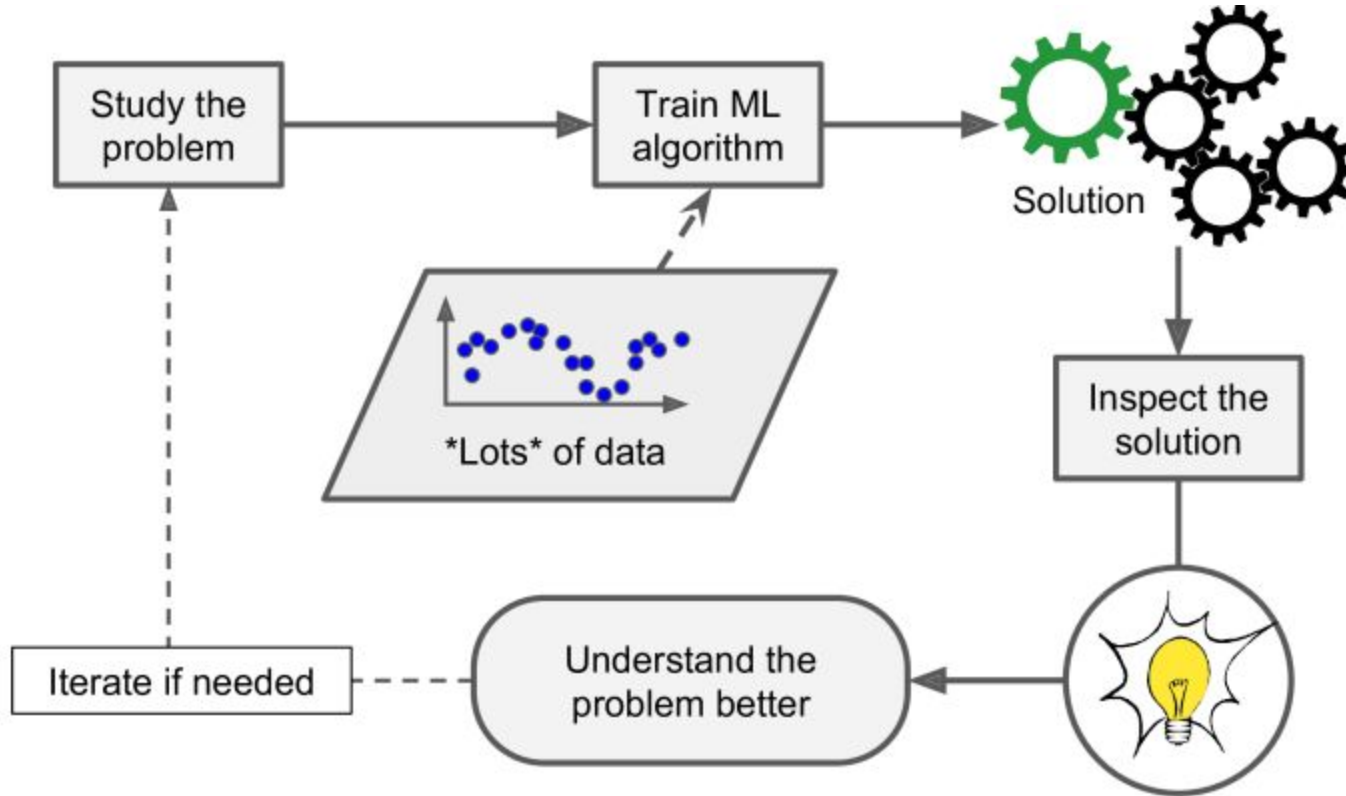
Since the problem is not trivial, your program will likely become a long list of complex rules—pretty hard to maintain.

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 . The program is much shorter, easier to maintain, and most likely more accurate.

Data

Update data

Launch!

Can be automated

Train ML algorithm

Evaluate solution

Study the problem → Train ML algorithm → Solution

Lots of data

Inspect the solution

Understand the problem better

Iterate if needed

To summarize, Machine Learning is great for:

- 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.
- Complex problems for which there is no good solution at all using a traditional approach: the best Machine Learning techniques can find a solution.
- Fluctuating environments: a Machine Learning system can adapt to new data.
- Getting insights about complex problems and large amounts of data.

Types of Machine Learning Systems

There are so many different types of Machine Learning systems that it is useful to classify them in board 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 batchlearning)
- 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-the-art 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.

Let's look at each of these criteria a bit more closely
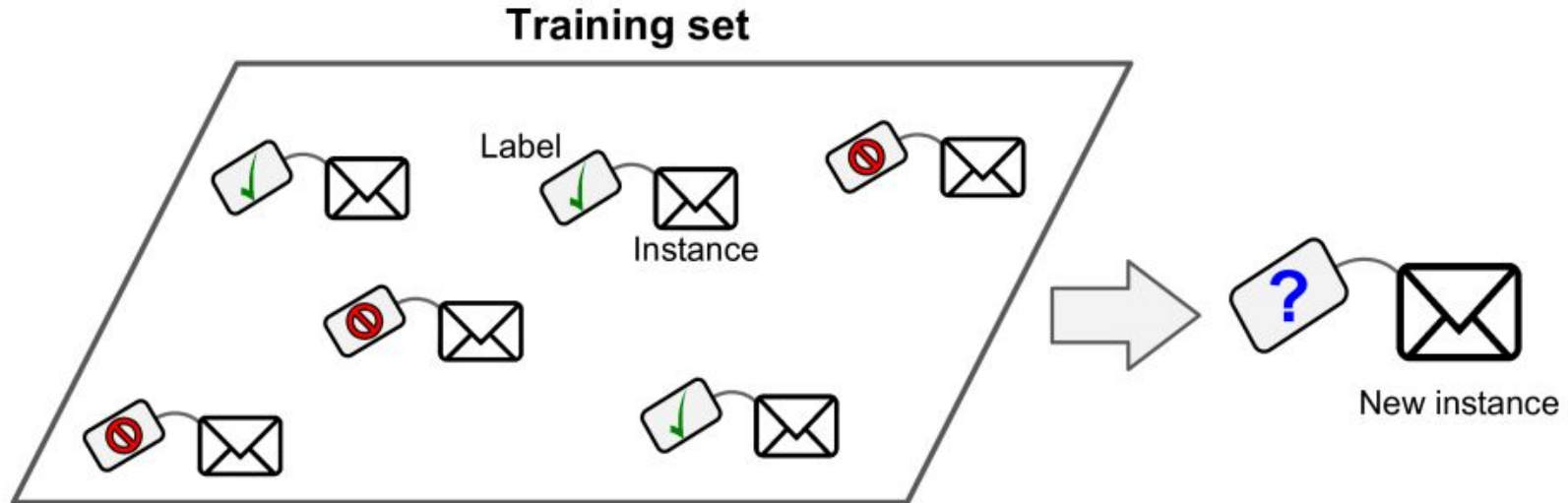
Supervised/Unsupervised Learning

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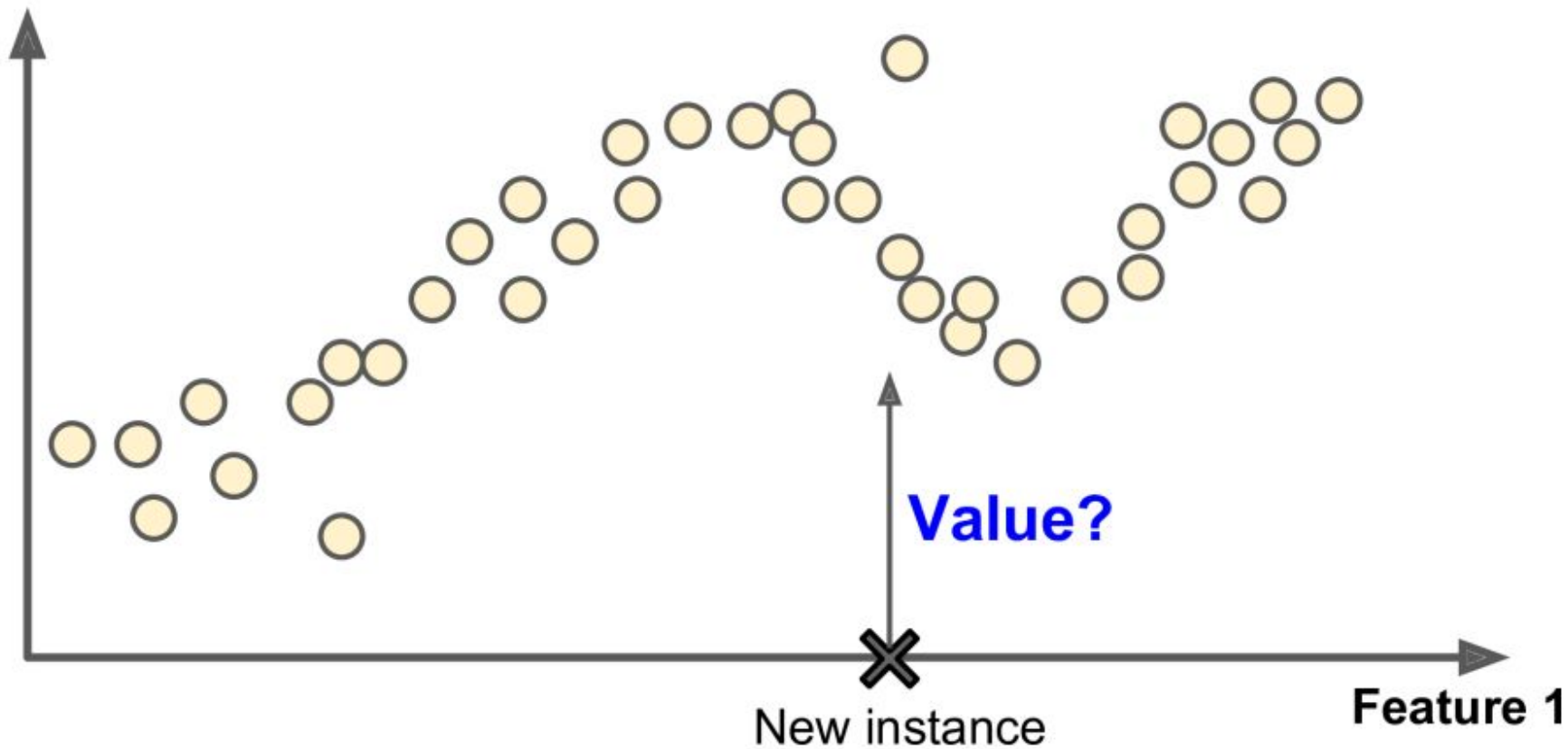
supervised learning, **unsupervised learning**, **semisupervised learning**, and **Reinforcement Learning.**

Supervised learning

In supervised learning, the training data you feed to the algorithm includes the desired solutions, called *labels*

Value

Value?

New instance

Feature 1

Here are some of the most important supervised learning algorithms:

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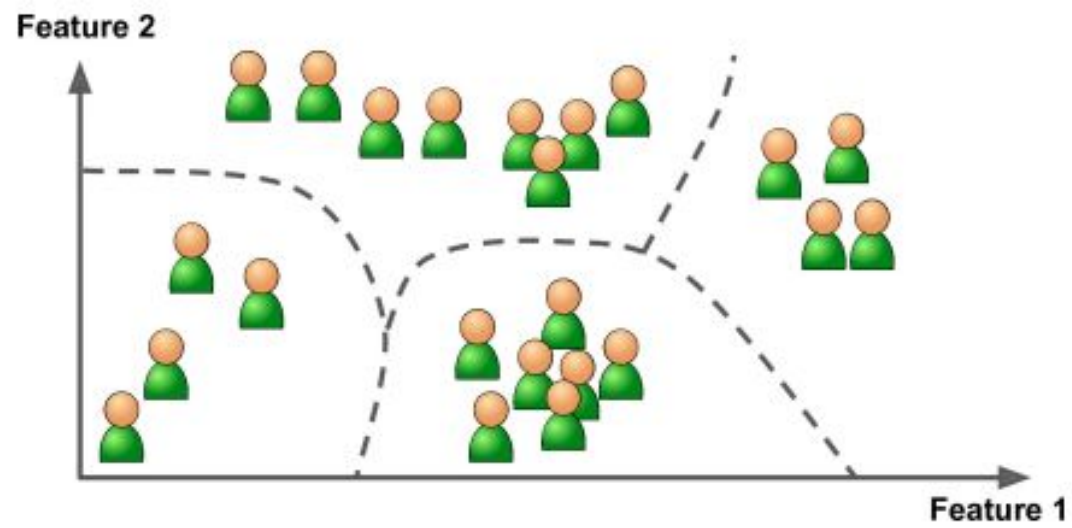- ❖ Decision Trees and Random Forests
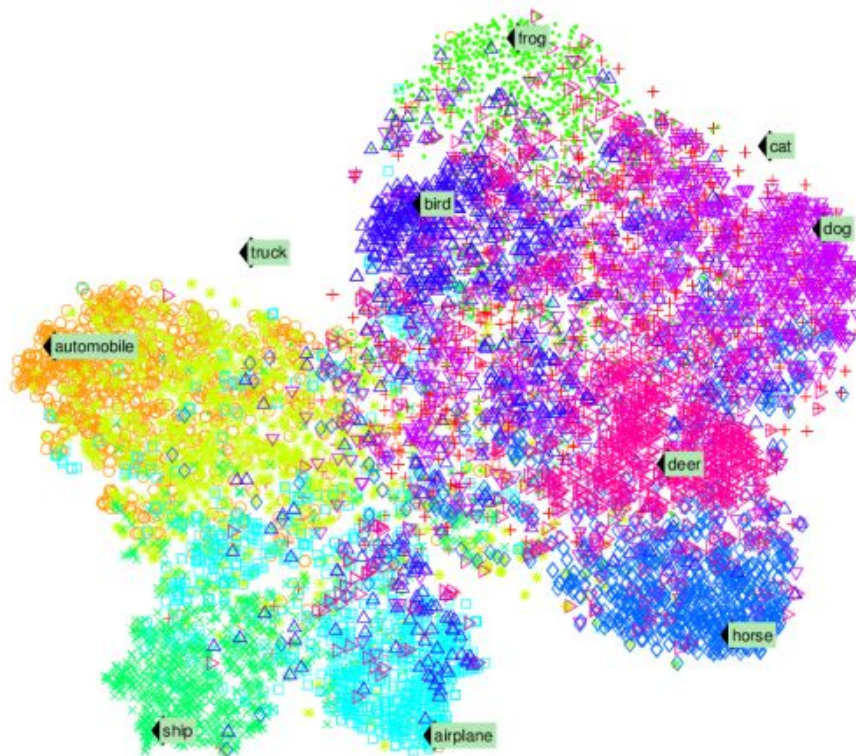- ❖ Neural networks

Unsupervised learning

In unsupervised learning, as you might guess, the training data is unlabeled The system tries to learn without a teacher.

Training set
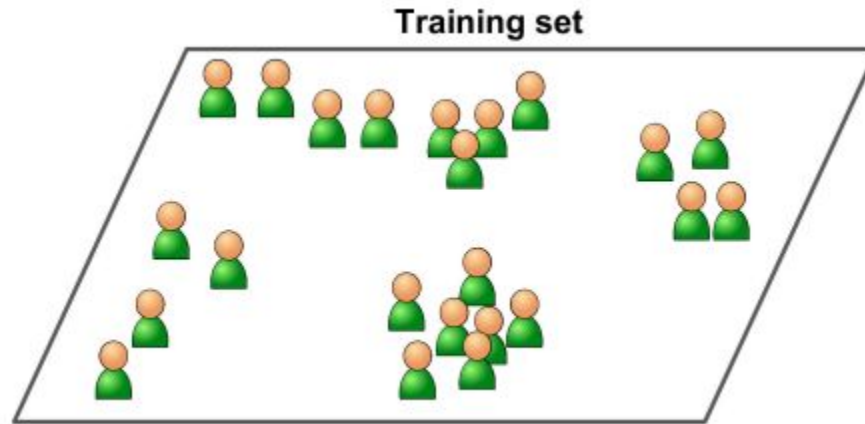
Here are some of the most important unsupervised learning algorithms:

- Clustering
  - K-Means
  - DBSCAN
  - Hierarchical Cluster Analysis (HCA)
- Anomaly detection and novelty detection
  - One-class SVM
  - Isolation Forest
- 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

Feature 2

Feature 1

+	cat
○	automobile
·	truck
·	frog
×	ship
□	airplane
◇	horse
△	bird
▽	dog
▷	deer

Feature 2

New instances

Anomaly ✖

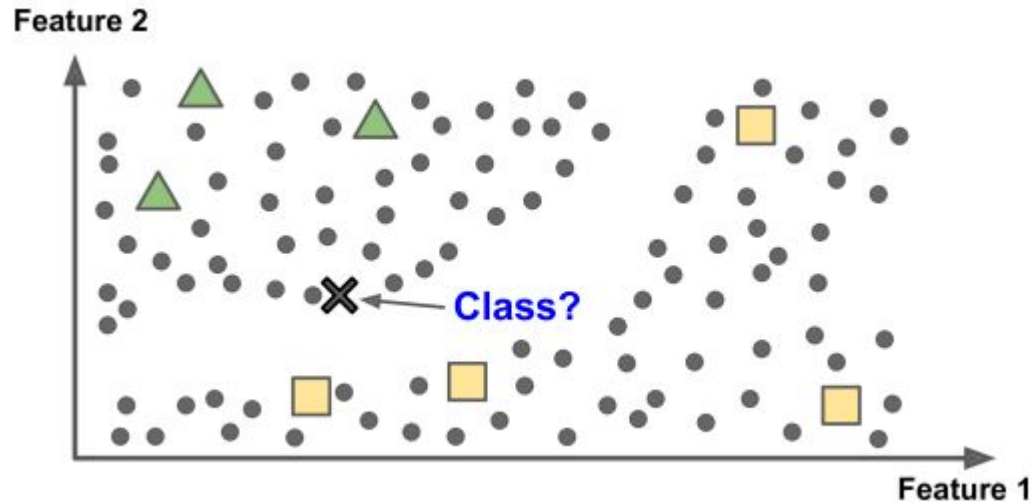Normal ✖

Training instances

Feature 1

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*

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
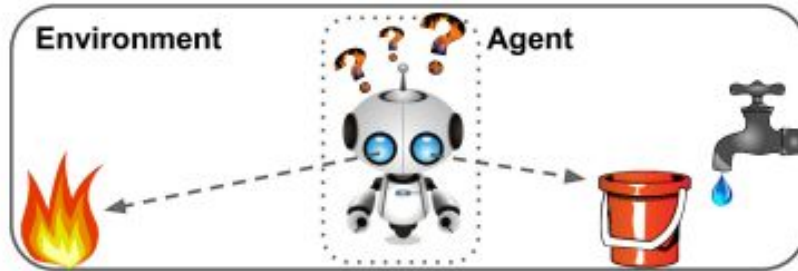
1 Observe

2 Select action using policy

Ouch!
-50 points

3 Action!

4 Get reward or penalty

= bad!
...
Next time avoid it.

5 Update policy (learning step)

6 Iterate until an optimal policy is found
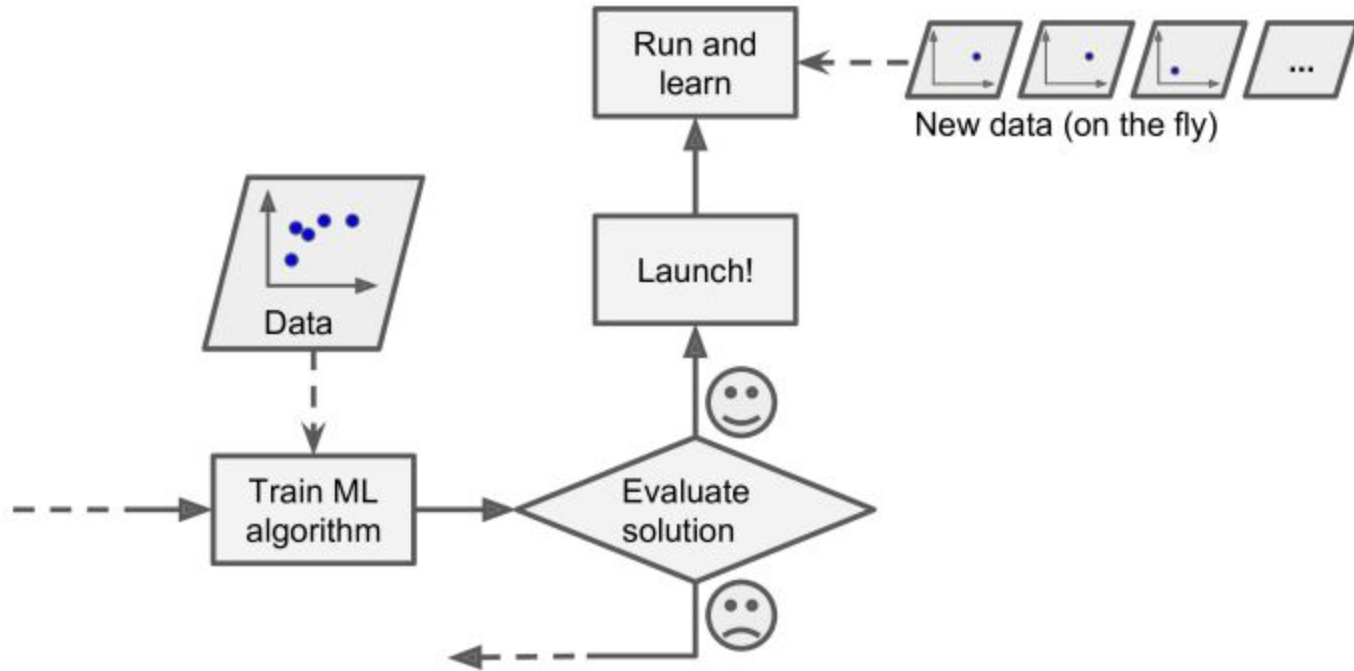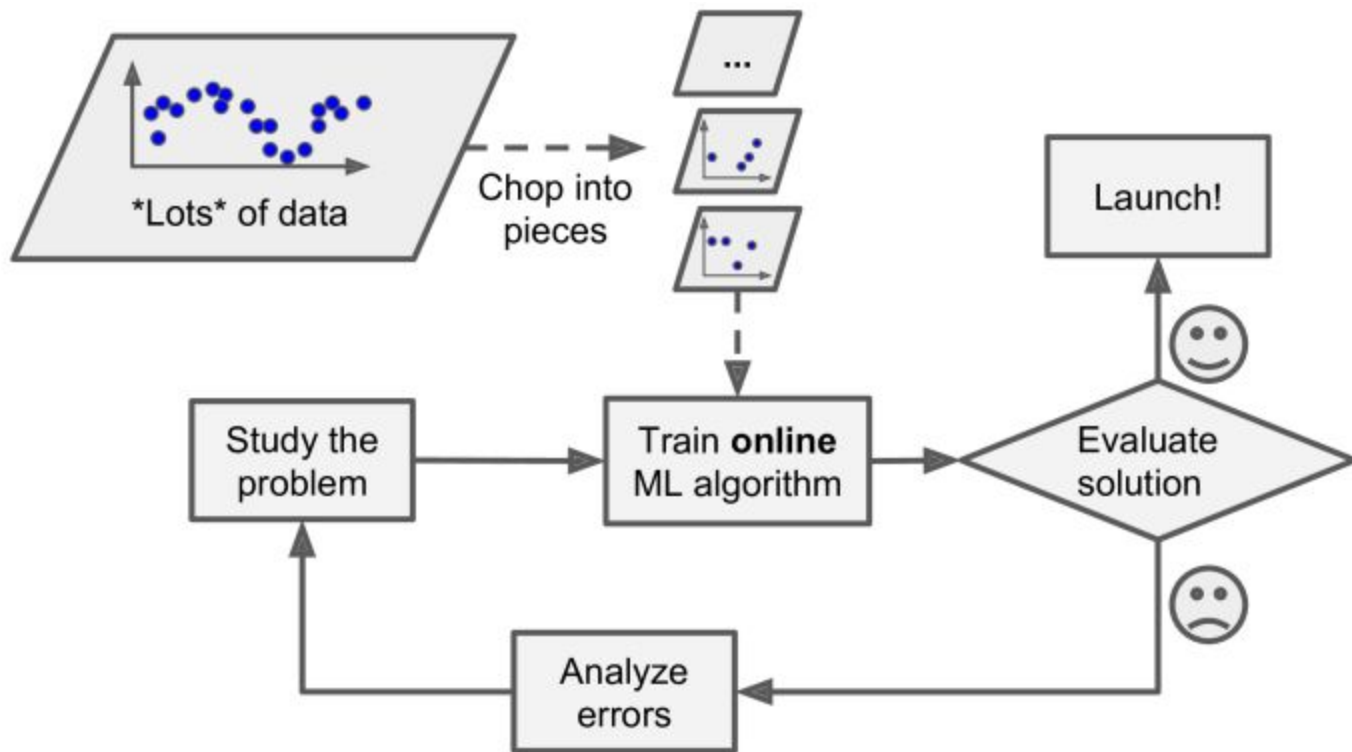
Batch and Online Learning

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.

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

Run and learn

New data (on the fly)

Launch!

Data

Train ML algorithm

Evaluate solution

Lots of data

Chop into pieces

...

Launch!

Study the problem

Train **online** ML algorithm

Evaluate solution

Analyze errors

Instance-Based Versus Model-Based Learning

Instance-Based Versus Model-Based Learning

## Instance-based learning

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

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

Feature 2

Training instances

New instance

Feature 1

Feature 2

Model

New instance

Feature 1
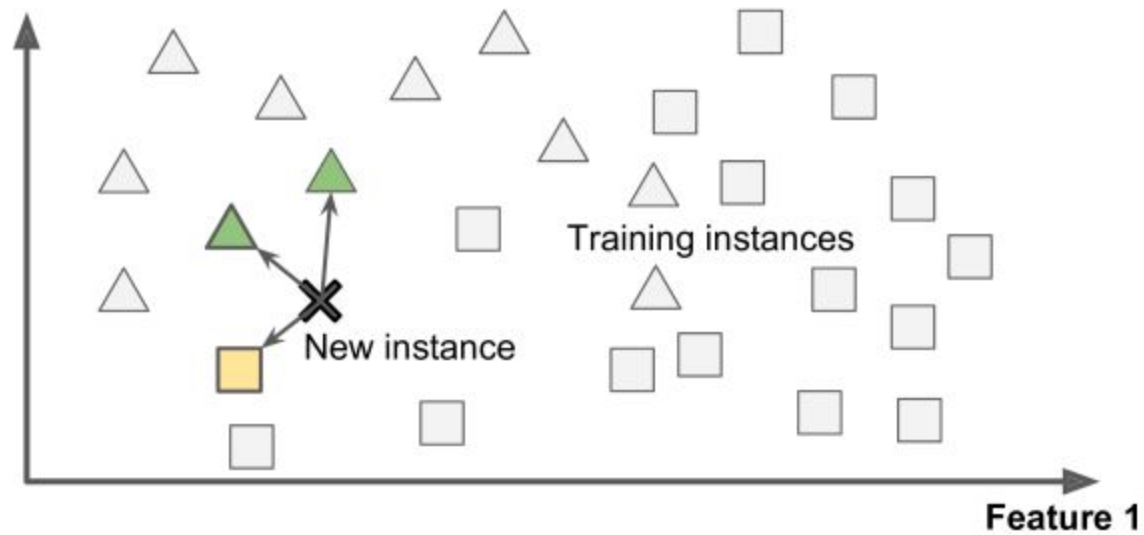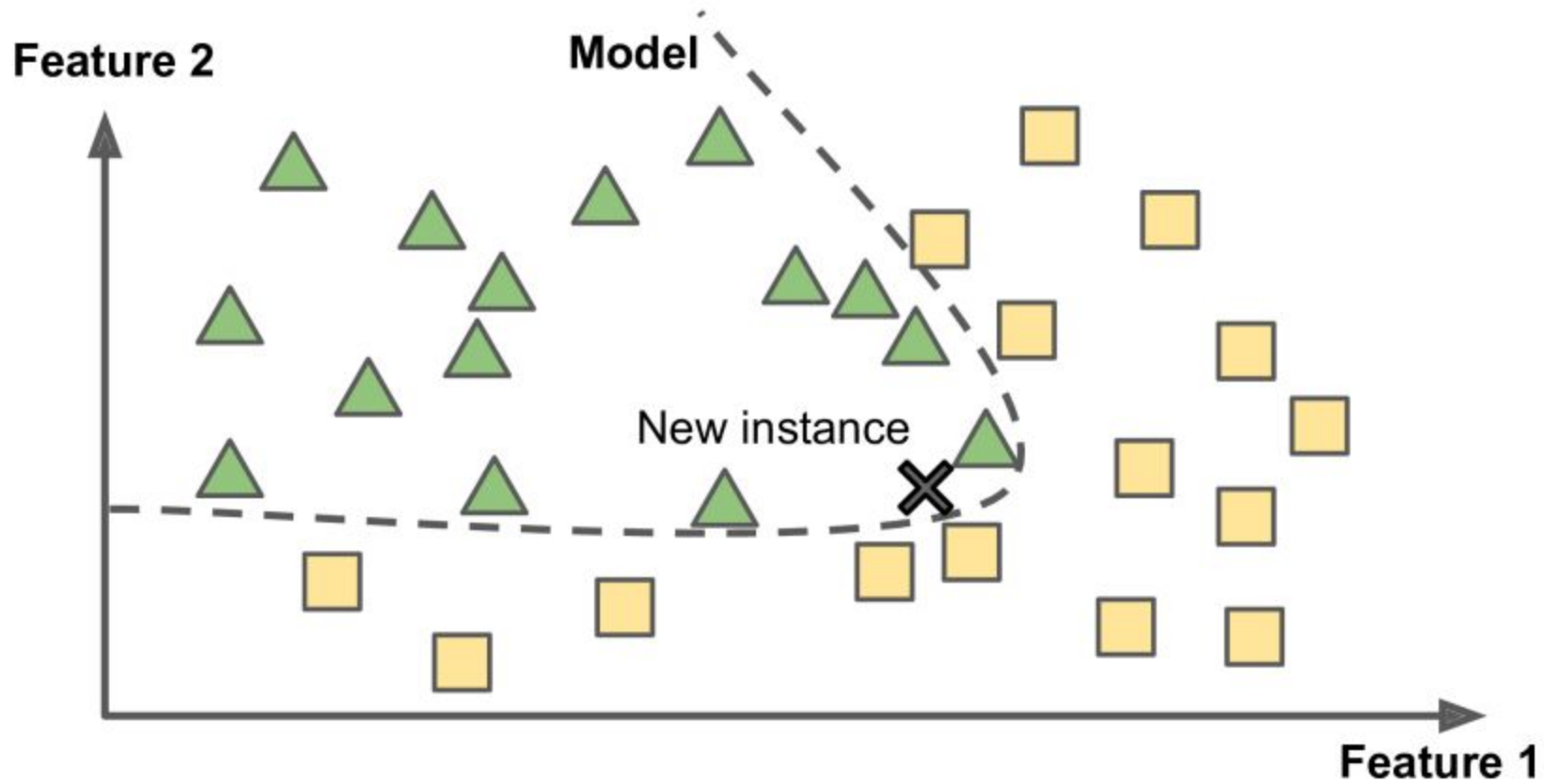
Country	GDP per capita (USD)	Life satisfaction
Hungary	12,240	4.9
Korea	27,195	5.8
France	37,675	6.5
Australia	50,962	7.3
United States	55,805	7.2

$\theta_0 = 8$
$\theta_1 = -5 \times 10^{-5}$

$\theta_0 = 4$
$\theta_1 = 5 \times 10^{-5}$

$\theta_0 = 0$
$\theta_1 = 2 \times 10^{-5}$

$$\theta_0 = 4.85$$
$$\theta_1 = 4.91 \times 10^{-5}$$
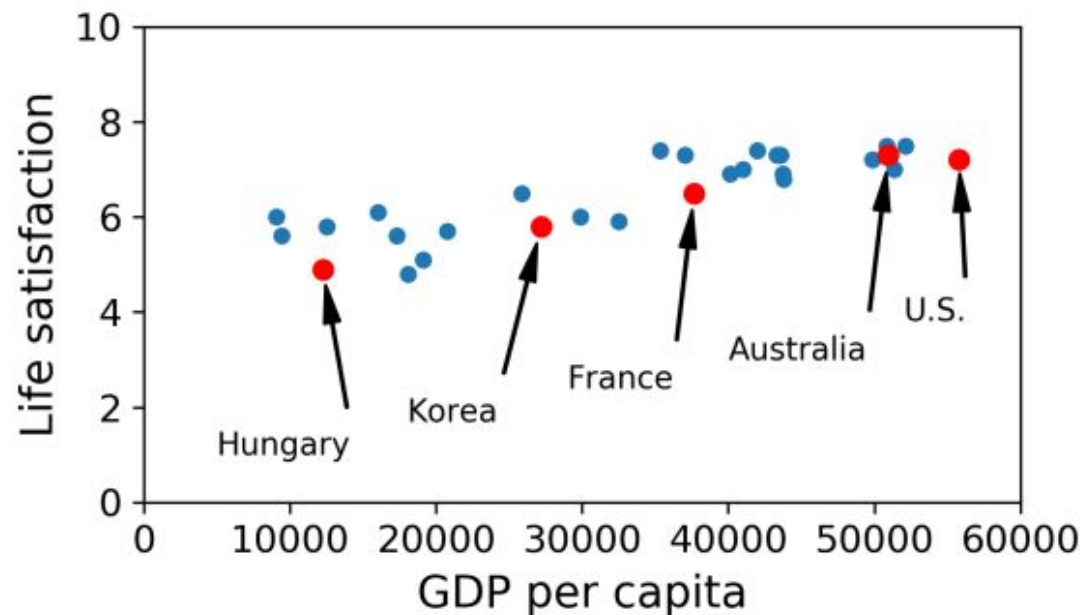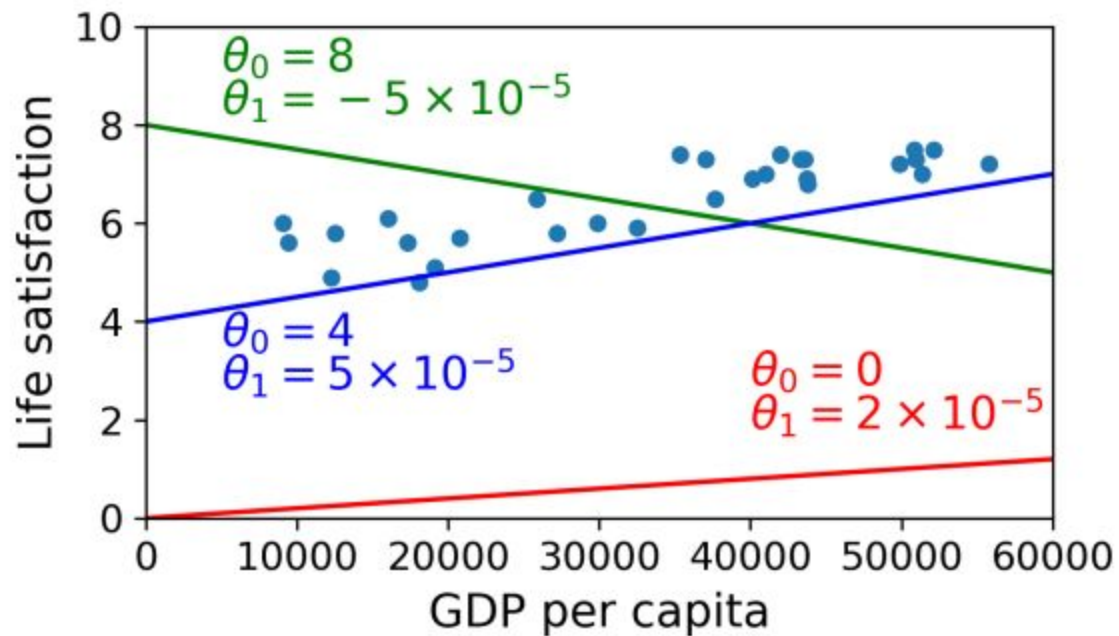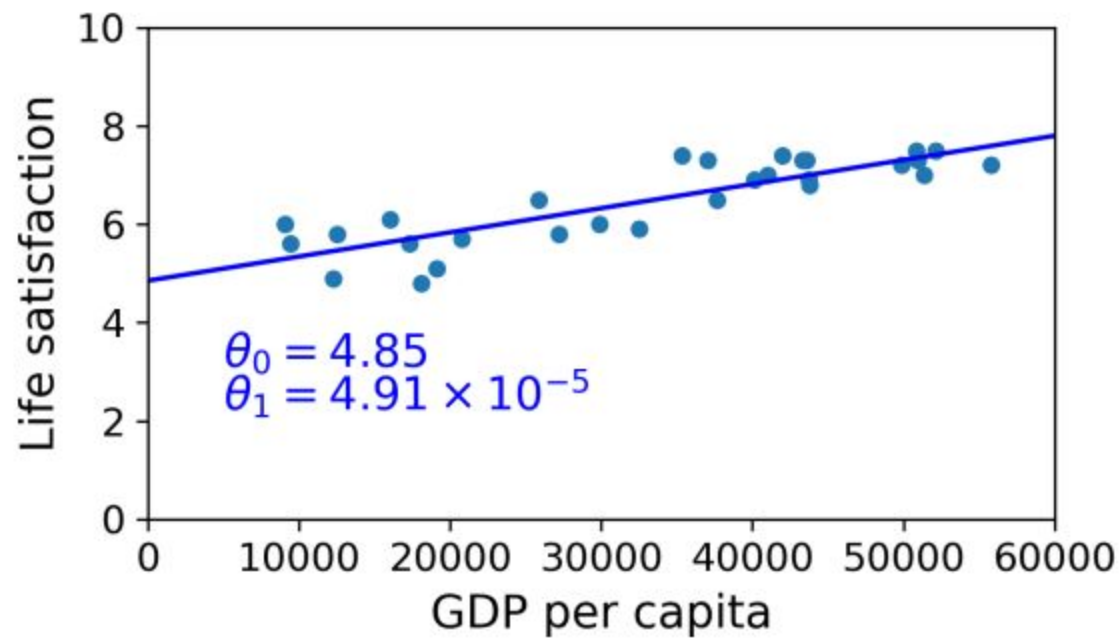
In summary:

- 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.

Main Challenges of Machine Learning

Main Challenges of Machine Learning

- Insufficient Quantity of Training Data
- Nonrepresentative Training Data
- Poor-Quality Data
- Irrelevant Features
- Overtting the Training Data

Testing and Validating

Hyperparameter Tuning and Model Selection

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 particular set. This means that the model is unlikely to perform as well on new data.

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 heldout 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 vali- dation process, you train the best model on the full training set (including the valida- tion 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? Well, since the final model will be trained on the full training set, it is not ideal to compare candidate models trained on a much smaller training set. 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, we get a much more accurate measure of its performance. However, there is a drawback: the training time is multiplied by the number of validation sets.

Data Mismatch

In some cases, it is easy to get a large amount of data for training, but it is not perfectly representative of the data that will be used in production. For example, suppose you want to create a mobile app to take pictures of flowers and automatically determine their species. You can easily download millions of pictures of flowers on the web, but they won't be perfectly representative of the pictures that will actually be taken using the app on a mobile device. Perhaps you only have 10,000 representative pictures (i.e., actually taken with the app). In this case, the most important rule to remember is that the validation set and the test 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 your 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 part of the training pictures (from the web) in yet another set that Andrew Ng calls 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, so if performs poorly on the validation set, the problem must come 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. Conversely, if the model performs poorly on the train-dev set, then the model must have overfit the training set, so you should try to simplify or regularize the model, get more training data and clean up the training data, as discussed earlier.