The Machine Learning Landscape

What Is Machine Learning?

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

- [Machine Learning is the] field of study that gives computers the ability to learn without being explicitly programmed.
- —Arthur Samuel, *1959*

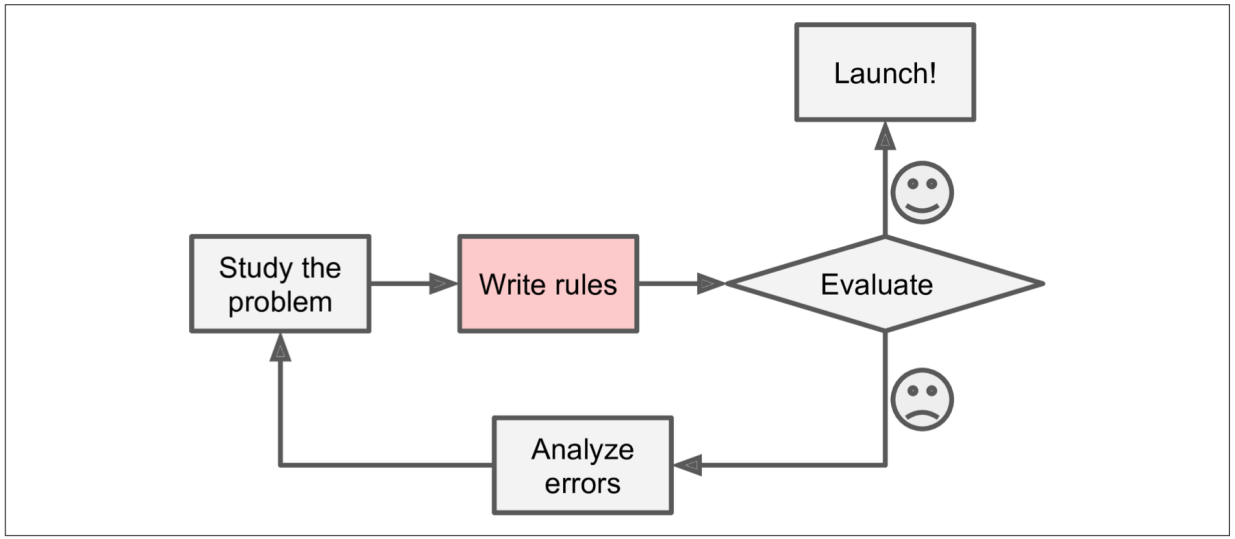
- And a more engineering-oriented one:
- A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

—Tom Mitchell, *1997*

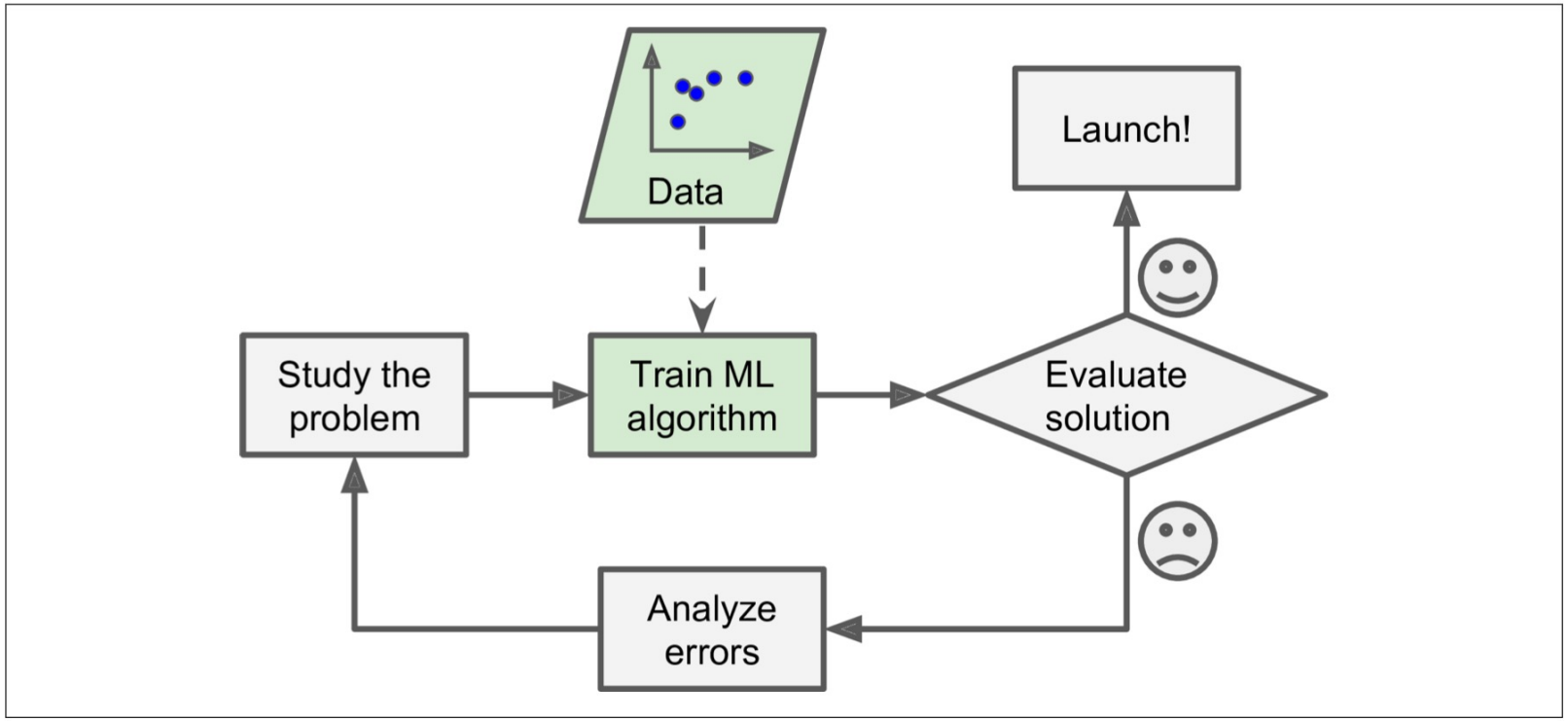
Why Use Machine Learning?

- 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.

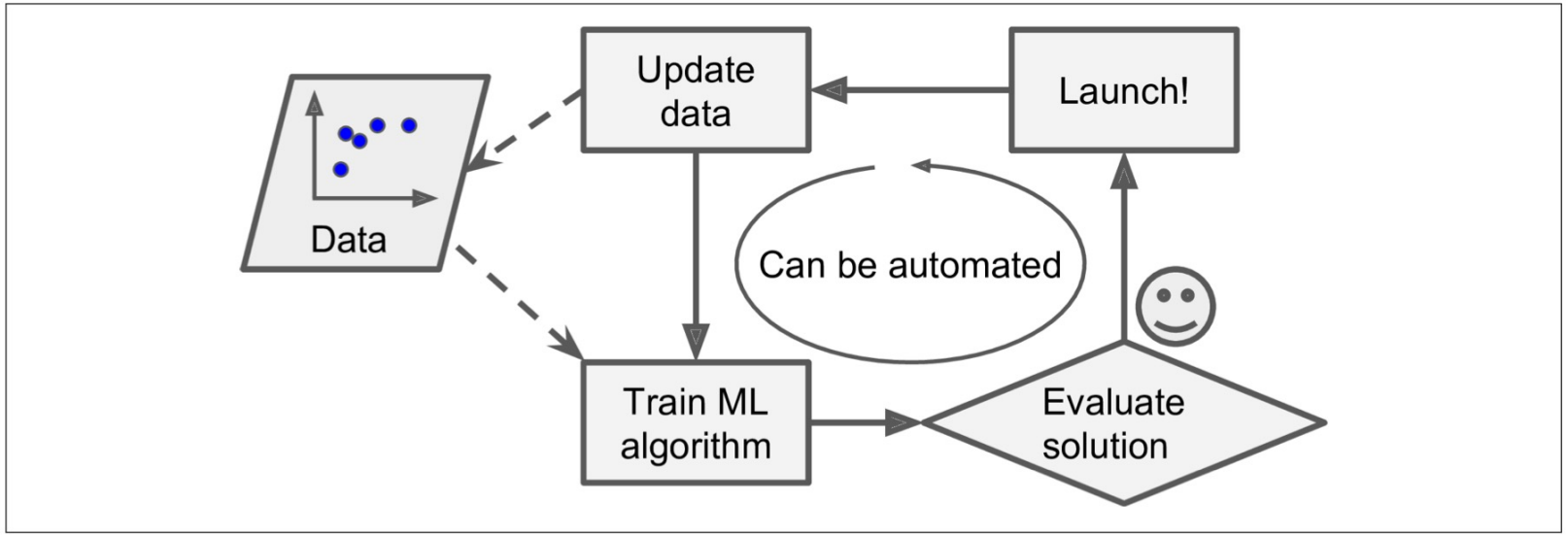


- Since the problem is not trivial, your program will likely become a long list of complex rules—pretty hard to maintain.
- In contrast, a spam filter based on Machine Learning techniques automatically learns which words and phrases are good predictors of spam by detecting unusually frequent patterns of words in the spam examples



Automatically adapting to change

- Moreover, if spammers notice that all their emails containing “4U” are blocked, they might start writing “For U” instead. A spam filter using traditional programming techniques would need to be updated to flag “For U” emails. If spammers keep working around your spam filter, you will need to keep writing new rules forever.
- In contrast, a spam filter based on Machine Learning techniques automatically notices that “For U” has become unusually frequent in spam flagged by users, and it starts flagging them without your intervention



- 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

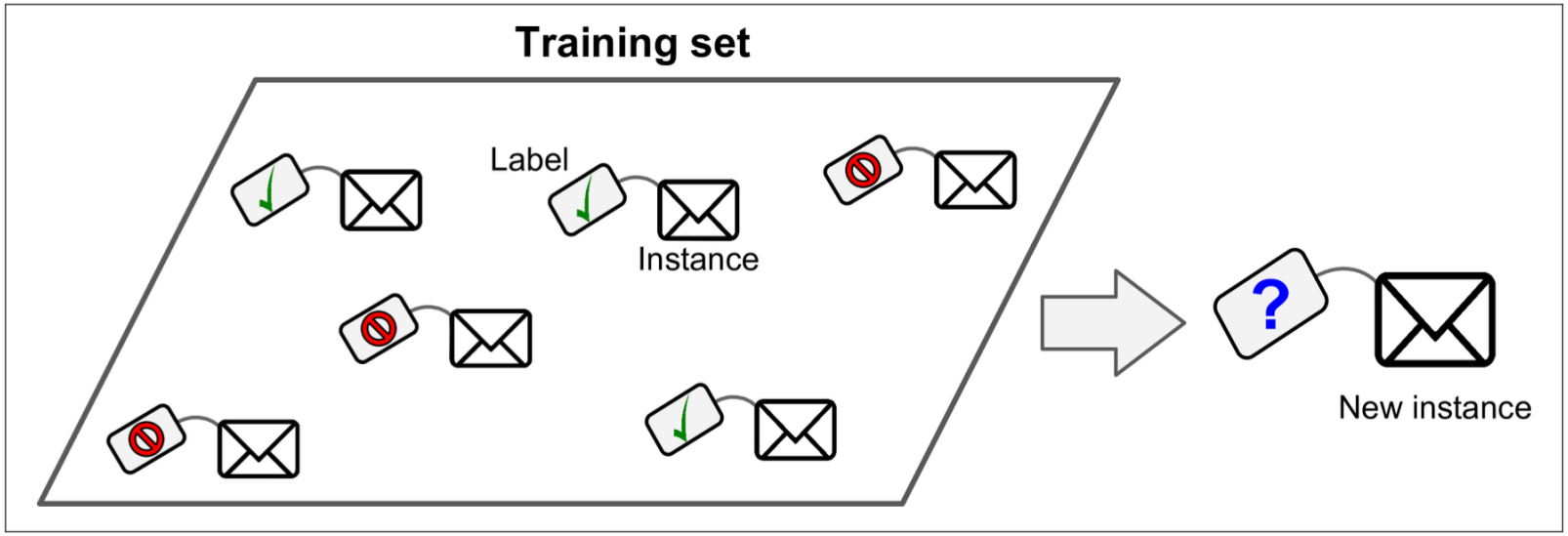
- There are so many different types of Machine Learning systems that it is useful to classify them in broad categories based on:
- Whether or not they are trained with human supervision (supervised, unsupervised, semi-supervised, and Reinforcement Learning)
- Whether or not they can learn incrementally 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)

Supervised/Unsupervised Learning

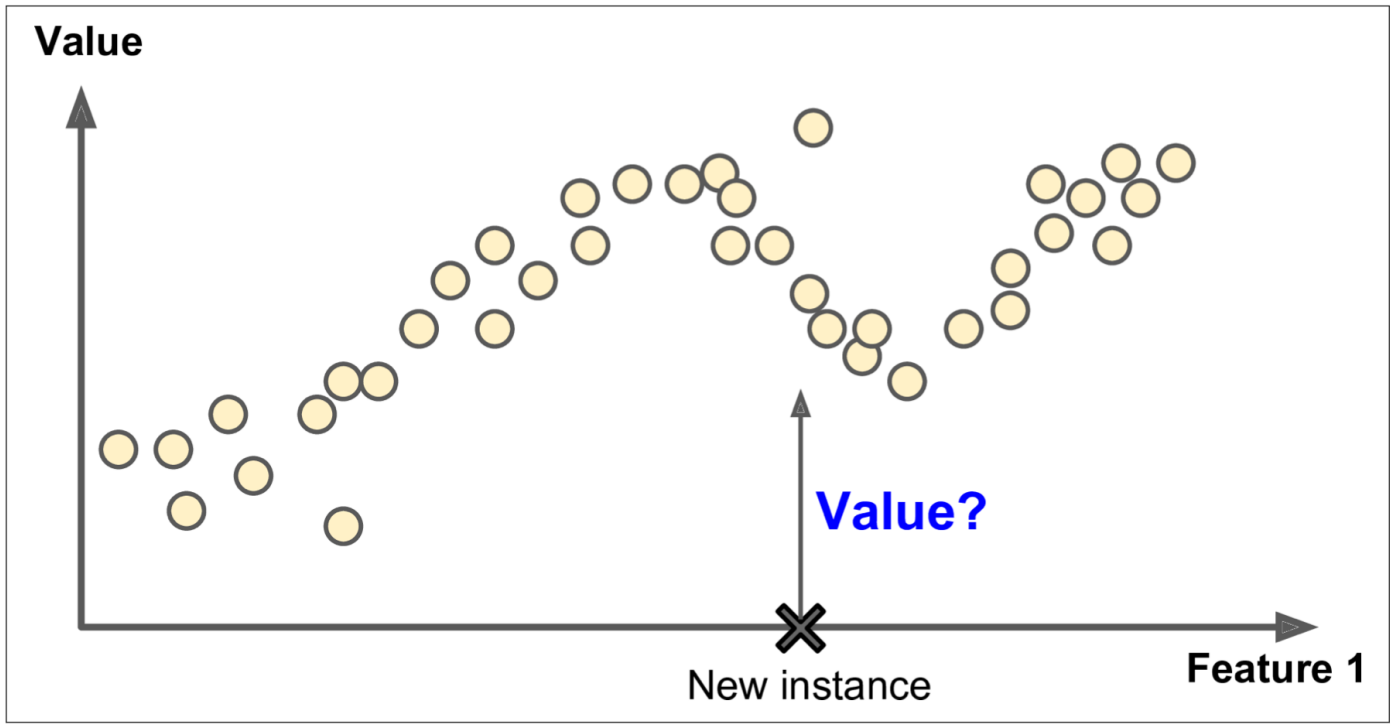
- Machine Learning systems can be classified according to the amount and type of supervision they get during training. There are four major categories: supervised learning, unsupervised learning, semi-supervised learning, and Reinforcement Learning.

1. Supervised learning

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



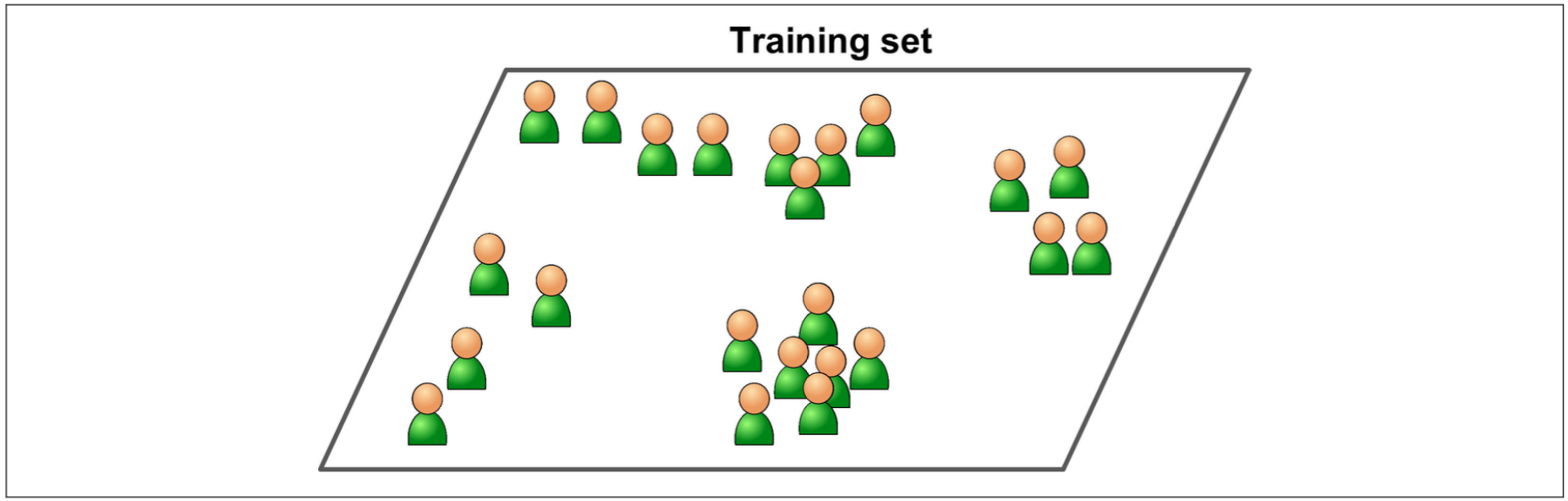
- A typical supervised learning task is *classification*. The spam filter is a good example of this: it is trained with many example emails along with their *class* (spam or ham), and it must learn how to classify new emails.
- Another typical task is to predict a *target* numeric value, such as the price of a car, given a set of *features* (mileage, age, brand, etc.) called *predictors*. This sort of task is called *regression*. To train the system, you need to give it many examples of cars, including both their predictors and their labels (i.e., their prices).



- Note that some regression algorithms can be used for classification as well, and vice versa. For example, *Logistic Regression* is commonly used for classification, as it can output a value that corresponds to the probability of belonging to a given class (e.g., 20% chance of being spam).
- 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.



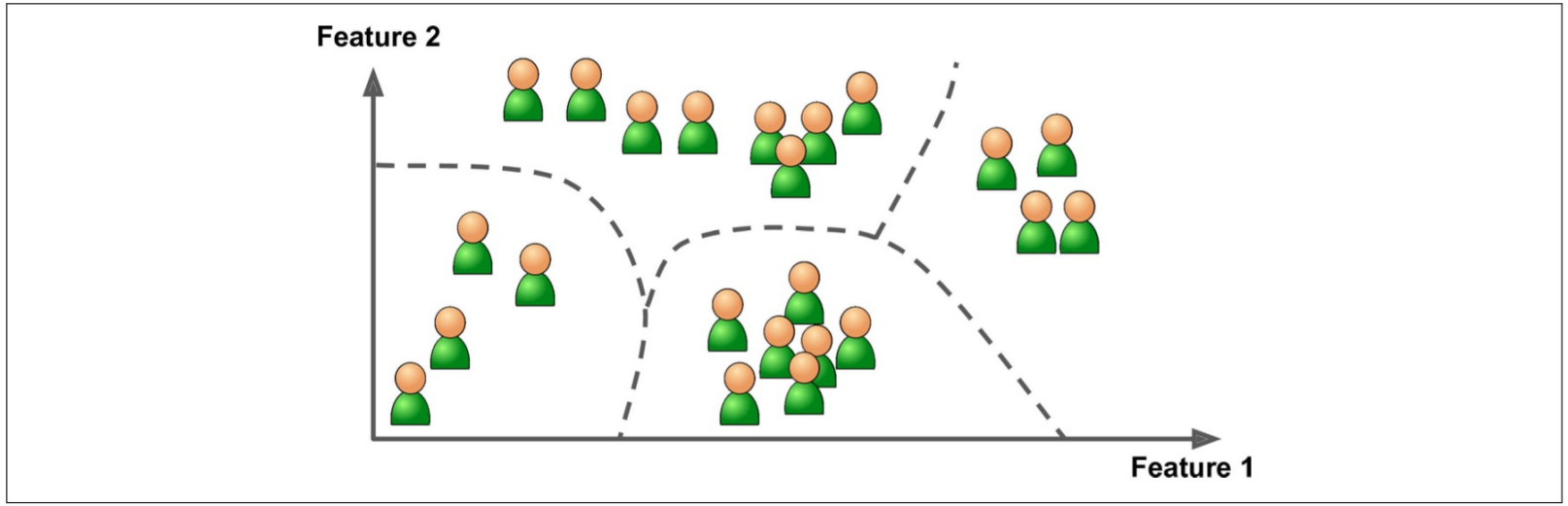
- Here are some of the most important unsupervised learning algorithms

1. Clustering
   — K-Means
   — DBSCAN
   — Hierarchical Cluster Analysis (HCA)

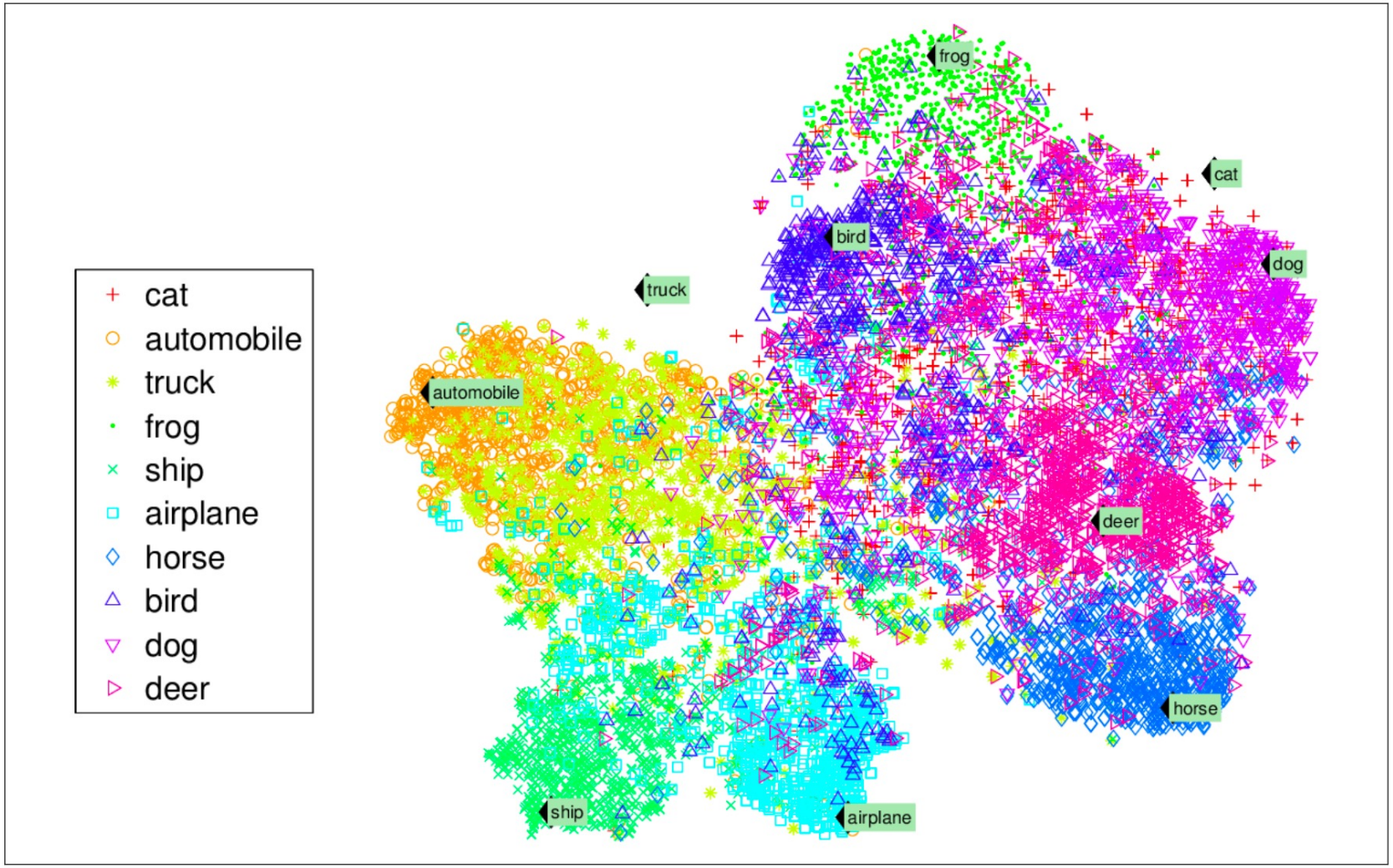
- • Anomaly detection and novelty detection
- — One-class SVM
  — Isolation Forest
- Visualization and dimensionality reduction
- — Principal Component Analysis (PCA) — Kernel PCA
  — Locally-Linear Embedding (LLE)
- — t-distributed Stochastic Neighbor Embedding (t-SNE)
- Association rule learning
- — Apriori
- — Eclat

Clustering

- 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. 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.

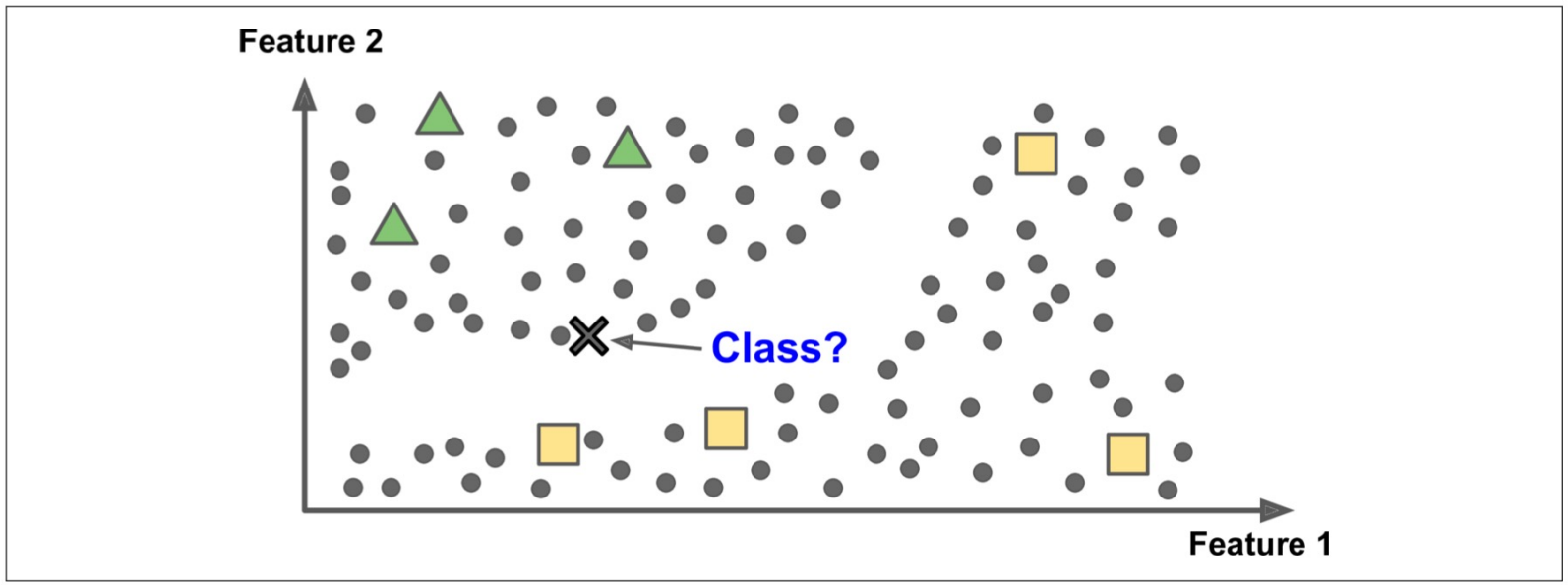


Visualization



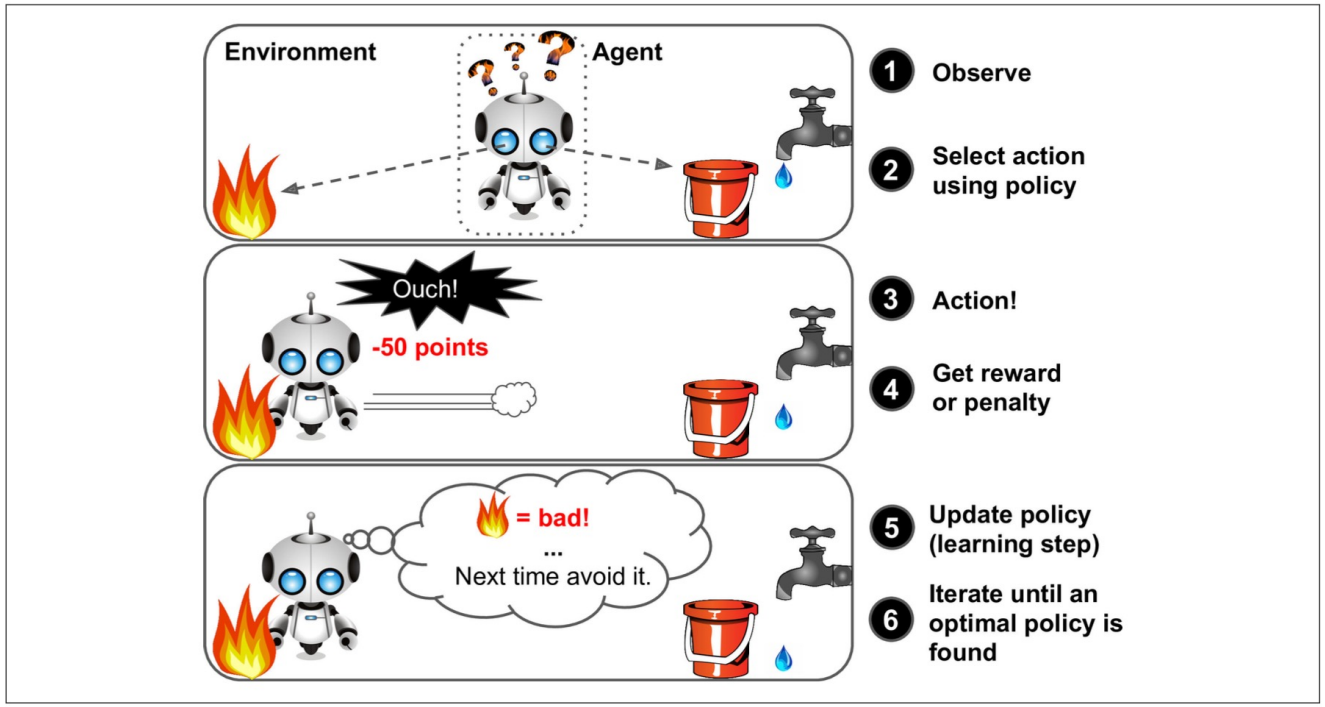
Semi-supervised learning

- 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 *semi-supervised 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.



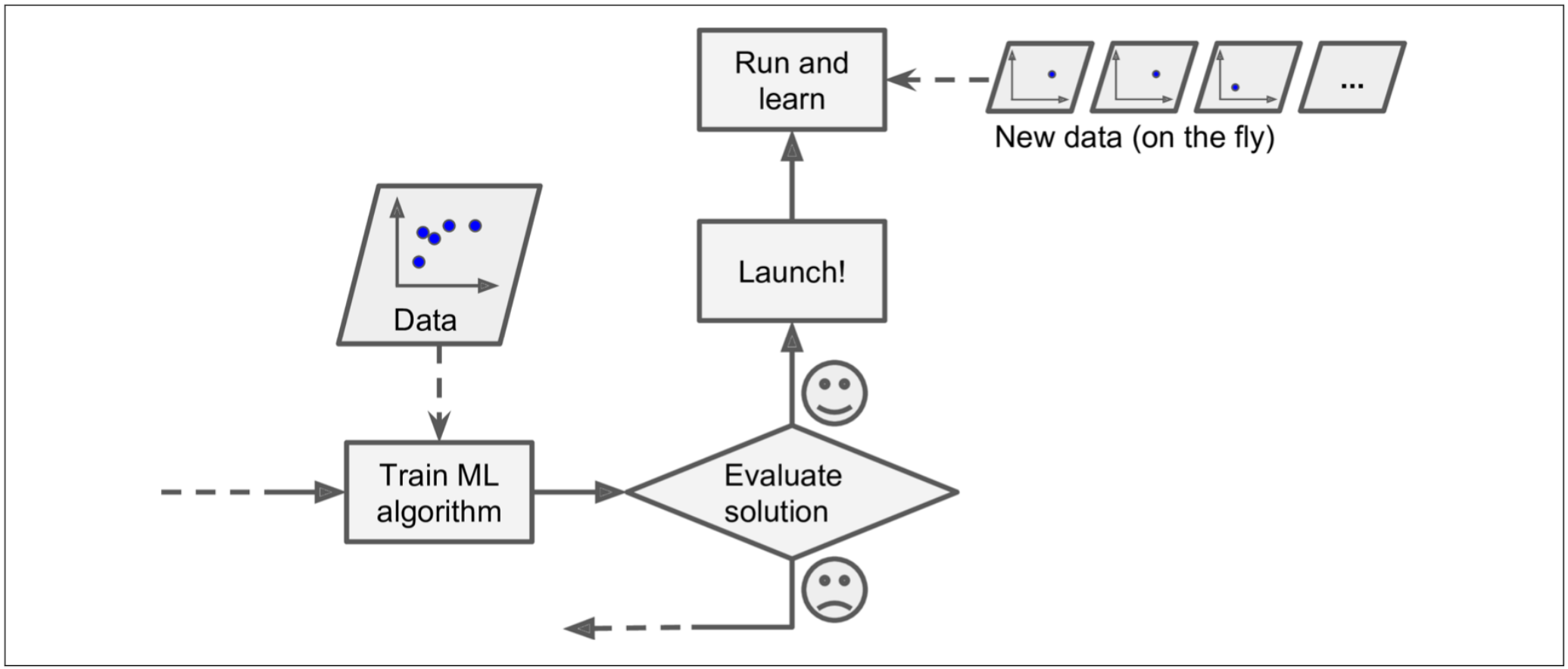
Reinforcement Learning

- *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.



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

Main Challenges of Machine Learning

- **Insufficient Quantity of Training Data**
- **Nonrepresentative Training Data**
- **Poor-Quality Data**
- **Irrelevant Features**
- **Overfitting the Training Data**
- **Underfitting the Training Data**

Stepping Back

By now you already know a lot about Machine Learning. However, we went through so many concepts that you may be feeling a little lost, so let's step back and look at the big picture:

- 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, and so on.
- In a 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 comparing them to the learned instances using a similarity measure.
- The system will not perform well if your training set is too small, or if the data is not representative, noisy, or 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).

Testing and Validating

1. 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. This works well, but if your model is horribly bad, your users will complain—not the best idea.
2. 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 estimate of this error. This value tells you how well your model will perform on instances it has never seen before.
3. 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.

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 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.