Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow

Concepts, Tools, and Techniques to Build Intelligent Systems

Stephen CUI[1]

October 30, 2022

[1] cuixuanStephen@gmail.com

Contents

Part I

The Fundamentals of Machine Learning

Chapter 1

The Machine Learning Landscape

Where does Machine Learning start and where does it end? What exactly does it mean for a machine to learn something? If I download a copy of Wikipedia, has my computer really learned something? Is it suddenly smarter? In this chapter we will start by clarifying what Machine Learning is and why you may want to use it.

Then, before we set out to explore the Machine Learning continent, we will take a look at the map and learn about the main regions and the most notable landmarks: supervised versus unsupervised learning, online versus batch learning, instance- based versus model-based learning. Then we will look at the workflow of a typical ML project, discuss the main challenges you may face, and cover how to evaluate and fine-tune a Machine Learning system.

This chapter introduces a lot of fundamental concepts (and jargon) that every data scientist should know by heart. It will be a high-level overview (it's the only chapter without much code), all rather simple, but you should make sure everything is crystal clear to you before continuing on to the rest of the book.

Tips: If you already know all the Machine Learning basics, you may want to skip directly to **?? ??**.

1.1 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

Your spam filter is a Machine Learning program that, given examples of spam emails (e.g., flagged by users) and examples of regular (nonspam, also called "ham") emails, can learn to flag spam. The examples that the system uses to learn are called the train- ing 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.

1.2 Why Use Machine Learning?

Machine Learning can help humans learn (**??**). ML algorithms can be inspected to see what they have learned (although for some algorithms this can be tricky). For instance, once a spam filter has been trained on enough spam, it can

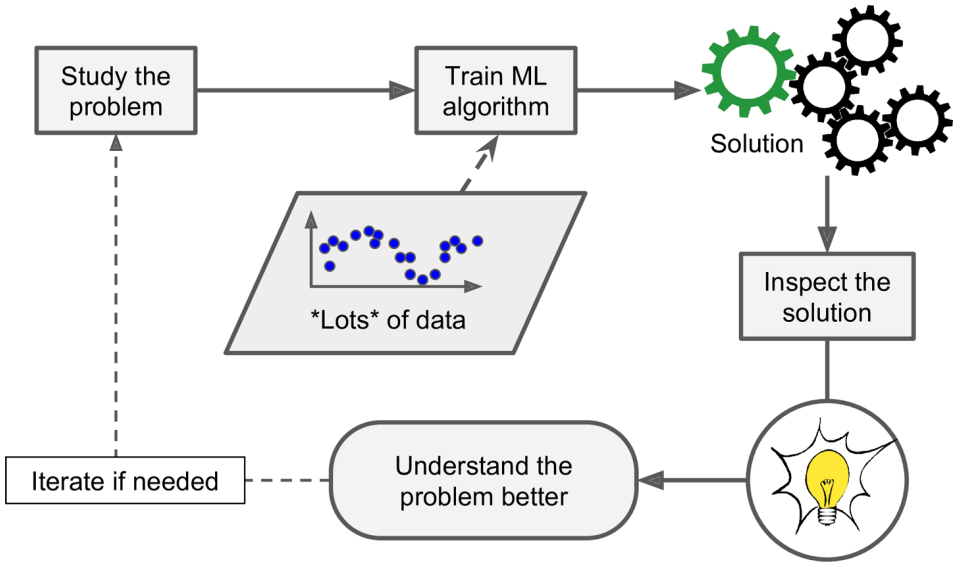

Figure 1.1: Machine Learning can help humans learn

easily be inspected to reveal the list of words and combinations of words that it believes are the best predictors of spam. Sometimes this will reveal unsuspected correlations or new trends, and thereby lead to a better understanding of the problem. Applying ML techniques to dig into large amounts of data can help discover patterns that were not immediately apparent. This is called data mining.

To summarize, 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 bet- ter than the traditional approach.
- Complex problems for which using a traditional approach yields no good solu- tion: 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.

1.3 Examples of Applications

- *Analyzing images of products on a production line to automatically classify them*

  This is image classification, typically performed using convolutional neural net- works (CNNs; see **??**).

- *Detecting tumors in brain scans*

  This is semantic segmentation, where each pixel in the image is classified (as we want to determine the exact location and shape of tumors), typically using CNNs as well.

- *Automatically classifying news articles*

  This is natural language processing (NLP), and more specifically text classifica- tion, which can be tackled using recurrent neural networks (RNNs), CNNs, or Transformers (see **??**).

- *Automatically flagging offensive comments on discussion forums*

  This is also text classification, using the same NLP tools.

- *Summarizing long documents automatically*

  This is a branch of NLP called text summarization, again using the same tools.

- *Creating a chatbot or a personal assistant*

  This involves many NLP components, including natural language understanding (NLU) and question-answering modules.

- *Forecasting your company's revenue next year, based on many performance metrics*

  This is a regression task (i.e., predicting values) that may be tackled using any regression model, such as a Linear Regression or Polynomial Regression model (see **??**), a regression SVM (see **??**), a regression Random Forest (see **??**), or an artificial neural network (see **??**). If you want to take into account sequences of past performance metrics, you may want to use RNNs, CNNs, or Transformers (see Chapters **??** and **??**).

- *Making your app react to voice commands*

  This is speech recognition, which requires processing audio samples: since they are long and complex sequences, they are typically processed using RNNs, CNNs, or Transformers (see Chapters **??** and **??**).

- *Detecting credit card fraud*

  This is anomaly detection (see **??**).

- *Segmenting clients based on their purchases so that you can design a different marketing strategy for each segment*

  This is clustering (see **??**).

- *Representing a complex, high-dimensional dataset in a clear and insightful diagram*

  This is data visualization, often involving dimensionality reduction techniques (see **??**).

- *Recommending a product that a client may be interested in, based on past purchases*

  This is a recommender system. One approach is to feed past purchases (and other information about the client) to an artificial neural network (see **??**), and get it to output the most likely next purchase. This neural net would typically be trained on past sequences of purchases across all clients.

- *Building an intelligent bot for a game*

  This is often tackled using Reinforcement Learning (RL; see **??**), which is a branch of Machine Learning that trains agents (such as bots) to pick the actions that will maximize their rewards over time (e.g., a bot may get a reward every time the player loses some life points), within a given environment (such as the game). The famous AlphaGo program that beat the world champion at the game of Go was built using RL.

1.4 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)

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, modelbased, supervised learning system.

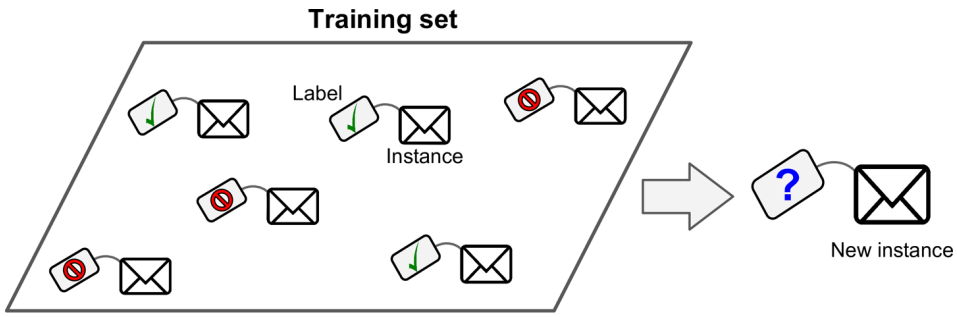

Figure 1.2: A labeled training set for spam classification

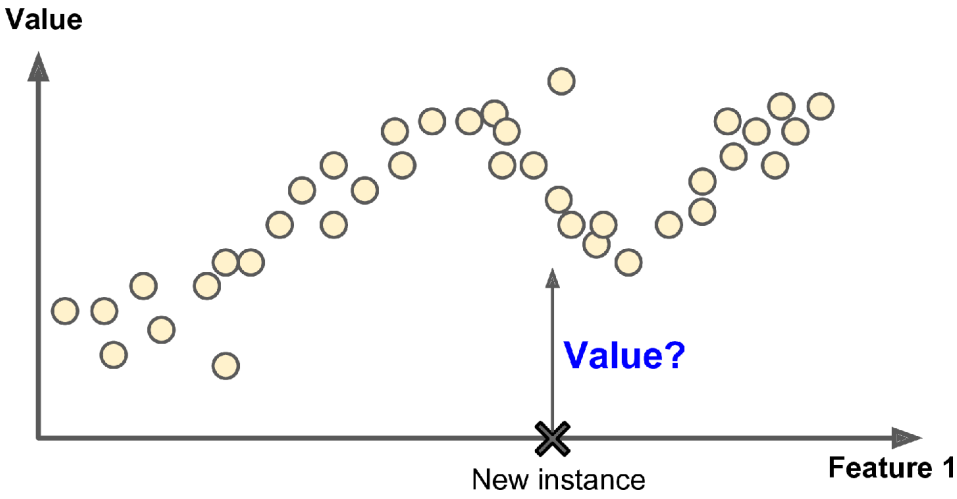

Figure 1.3: A regression problem: predict a value, given an input feature (there are usu- ally multiple input features, and sometimes multiple output values)

1.4.1 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 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 (**??**)[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).

Notes: 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.

Here are some of the most important supervised learning algorithms (covered in this book):

- k-Nearest Neighbors
- Linear Regression

[1] Fun fact: this odd-sounding name is a statistics term introduced by Francis Galton while he was studying the fact that the children of tall people tend to be shorter than their parents. Since the children were shorter, he called this regression to the mean. This name was then applied to the methods he used to analyze correlations between variables.

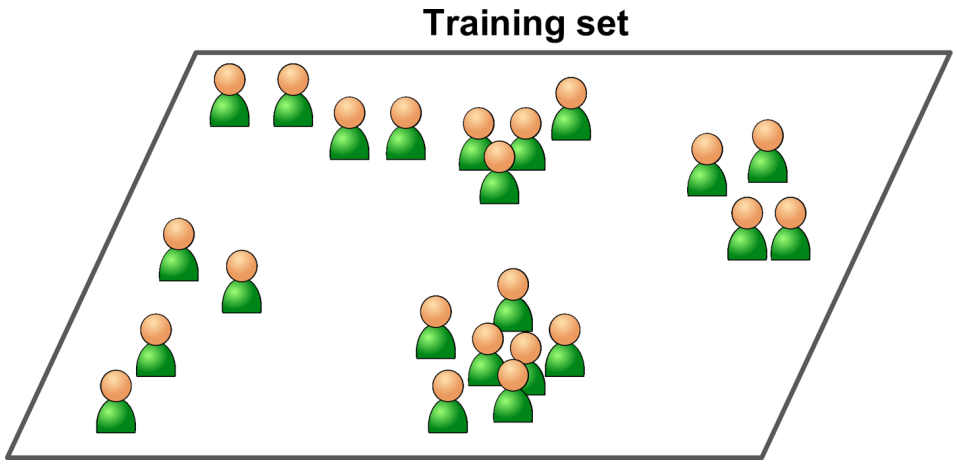

Figure 1.4: An unlabeled training set for unsupervised learning

- Logistic Regression
- Support Vector Machines (SVMs)
- Decision Trees and Random Forests
- Neural networks[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 (most of these are covered in Chapters **??** and **??**):

- 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

[2] 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

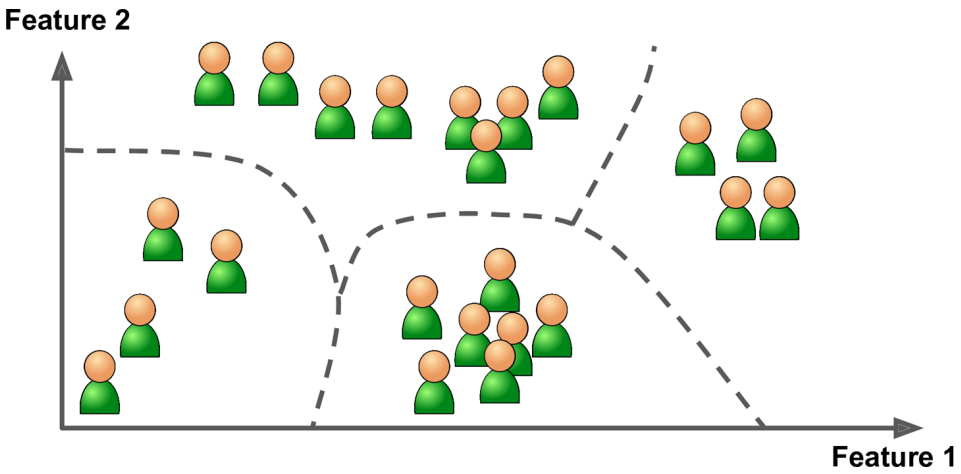

Figure 1.5: Clustering

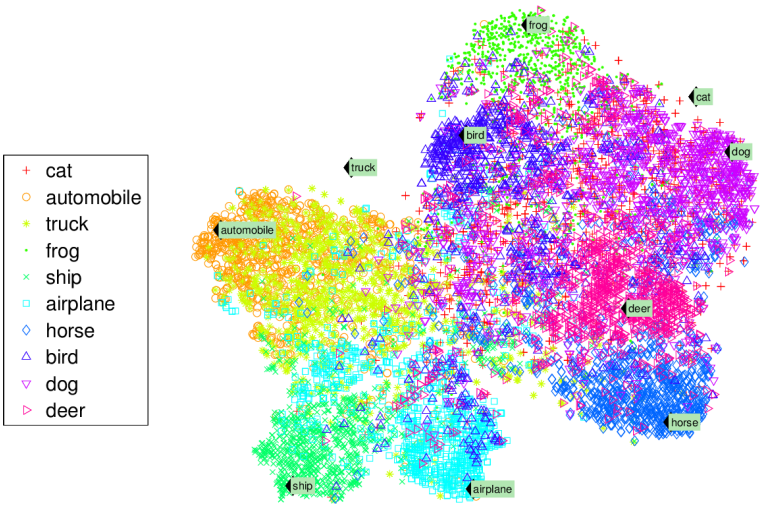

Figure 1.6: Example of a t-SNE visualization highlighting semantic clusters

– 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 (**??**). At no point do you tell the algorithm which group a visitor belongs to: it finds those connections without your help.

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 rep- resentation of your data that can easily be plotted (**??**[3]). 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.

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 correla- ted features into one. For example, a car's mileage may be strongly 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.

[3]Notice how animals are rather well separated from vehicles and how horses are close to deer but far from birds. Figure reproduced with permission from Richard Socher et al., "Zero-Shot Learning Through CrossModal Transfer," Proceedings of the 26th International Conference on Neural Information Processing Systems 1 (2013): 935–943.

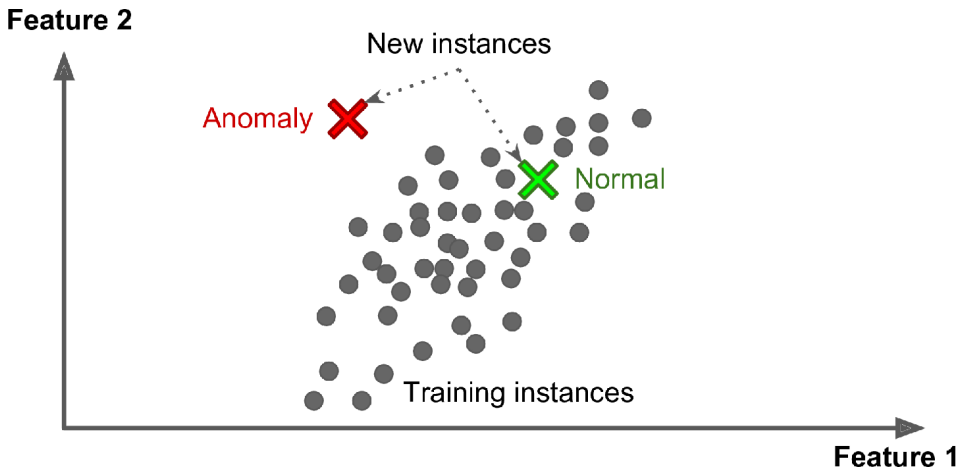

Figure 1.7: Anomaly detection

Notes: It is often a good idea to try to reduce the dimension of your train- ing data using a dimensionality reduction algorithm before you feed it to another Machine Learning algorithm (such as a super- vised learning algorithm). It will run much faster, the data will take up less disk and memory space, and in some cases it may also per- form better.

Yet another important unsupervised task is anomaly detection—for example, detect- ing unusual credit card transactions to prevent fraud, catching manufacturing defects, or automatically removing outliers from a dataset before feeding it to another learn- ing 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 (see **??**). 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. For example, if you have thousands of pictures of dogs, and 1% of these pictures represent Chihuahuas, then a novelty detection algorithm should not treat new pictures of Chihuahuas as novelties. On the other hand, anomaly detection algorithms may consider these dogs as so rare and so different from other dogs that they would likely classify them as anomalies (no offense to Chihuahuas).

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 one another.

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 (**??**).

Most semisupervised learning algorithms are combinations of unsupervised and supervised algorithms. For example, deep belief networks (DBNs) are based on unsu- pervised 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 shown in **??**). 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 May 2017 when it beat the world champion Ke Jie at the game of Go. It learned its winning policy by analyzing millions of games,

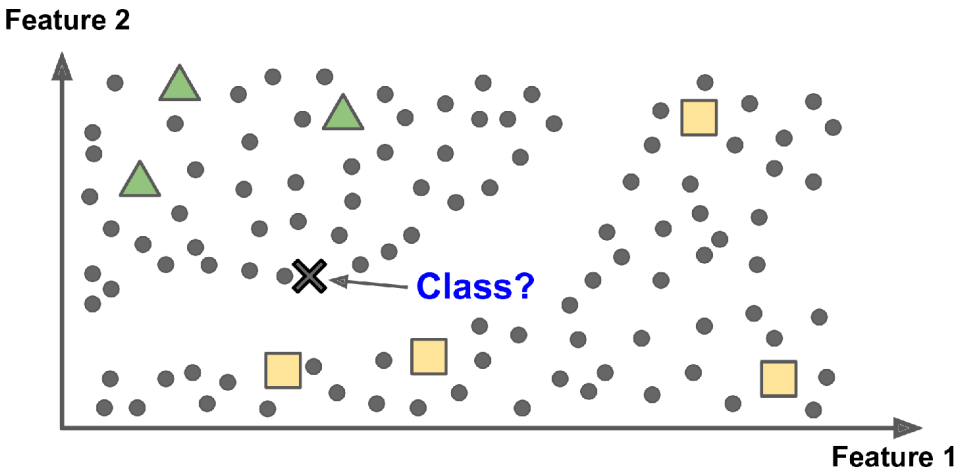

Figure 1.8: Semisupervised learning with two classes (triangles and squares): the unlabeled examples (circles) help classify a new instance (the cross) into the triangle class rather than the square class, even though it is closer to the labeled squares

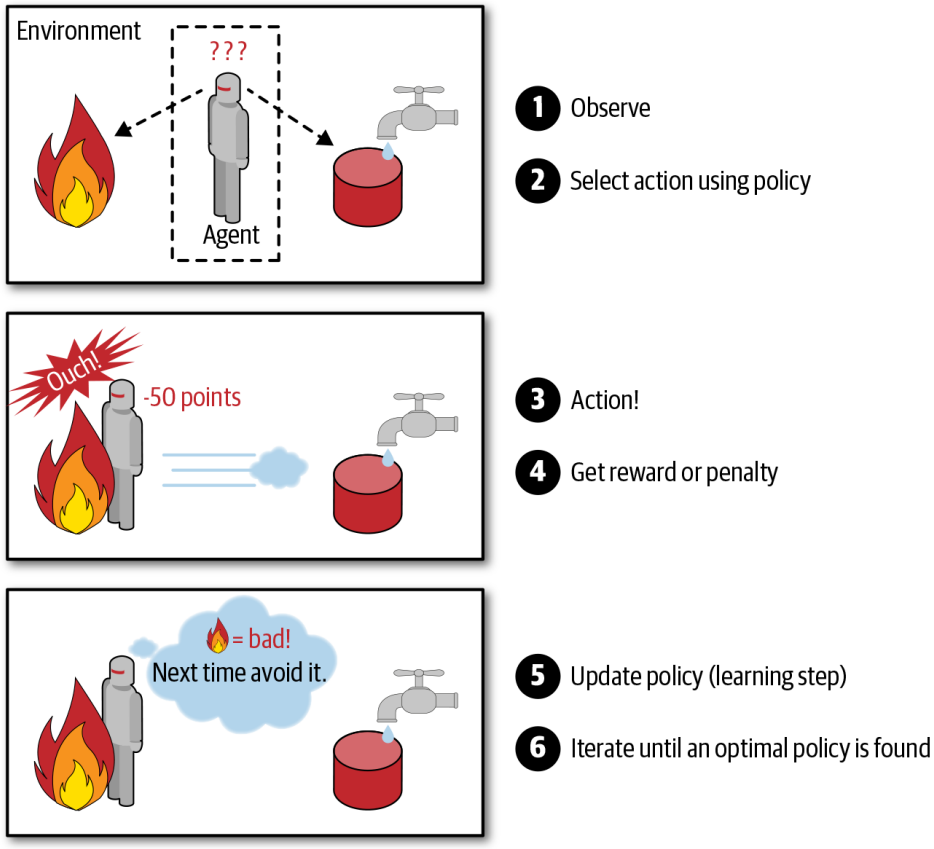

Figure 1.9: Reinforcement Learning

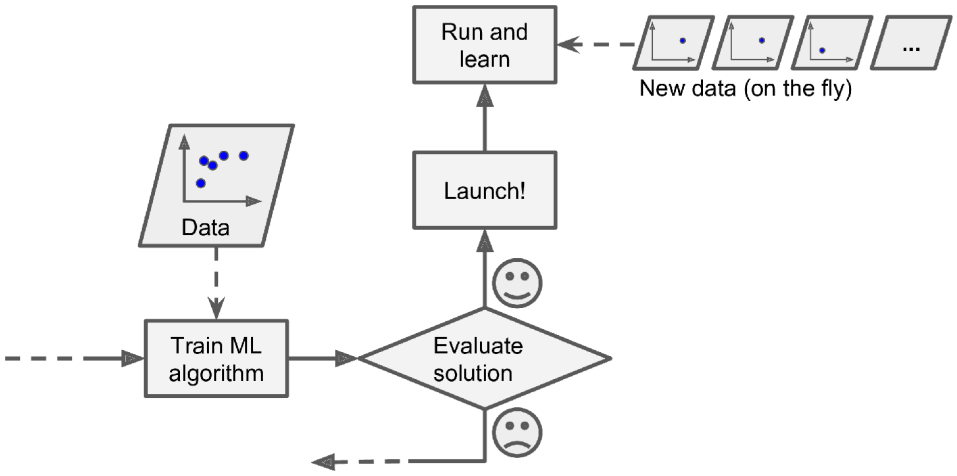

Figure 1.10: In online learning, a model is trained and launched into production, and then it keeps learning as new data comes in

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.

1.4.2 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.

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.

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.

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 (see **??**).

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 **??**).

Cautions: 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.

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. For example, bad data could come from a malfunctioning sensor

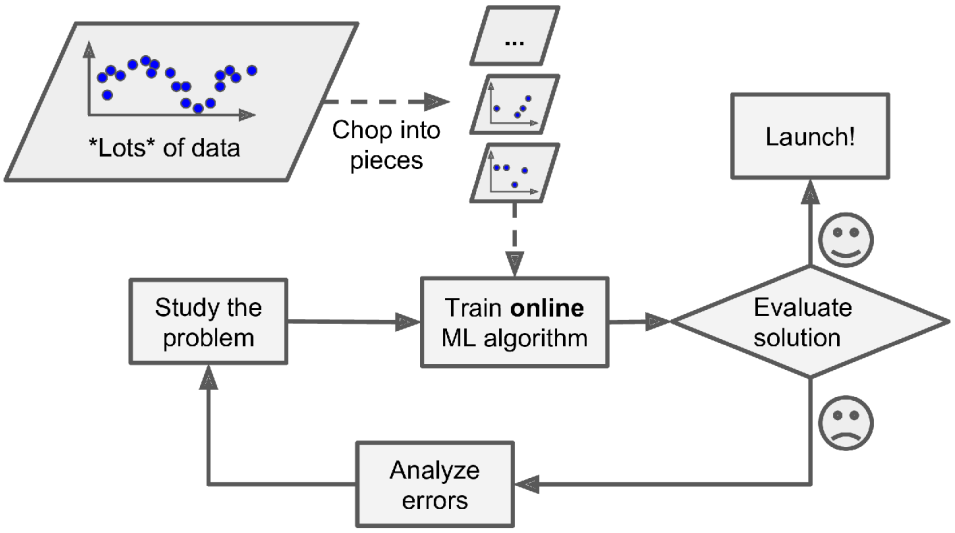

Figure 1.11: Using online learning to handle huge datasets

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).

1.4.3 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.

There are two main approaches to generalization: instance-based learning and 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.

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 by using a similarity measure to compare them to the learned examples (or a subset of them). For example, in **??** the new instance would be classified as a triangle because the majority of the most similar instances belong to that class.

Model-based learning

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

For example, suppose you want to know if money makes people happy, so you download the Better Life Index data from the OECD's website and stats about gross domestic product (GDP) per capita from the IMF's website.

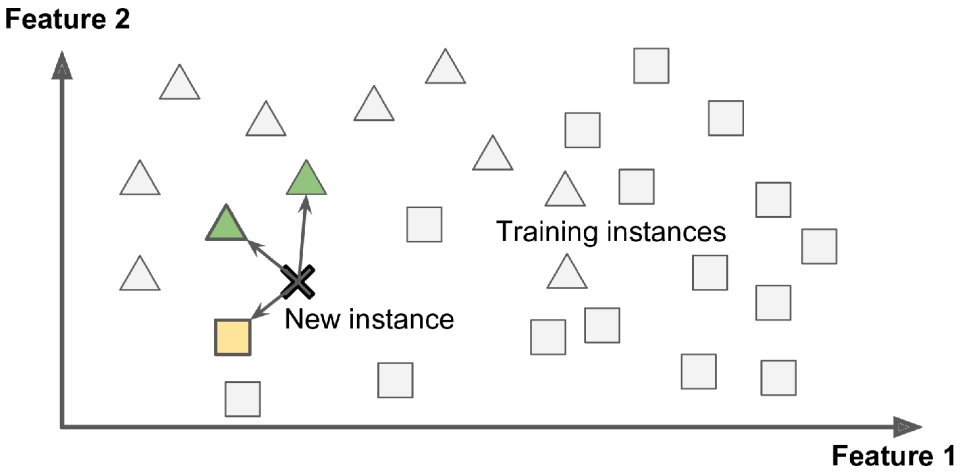

Figure 1.12: Instance-based learning

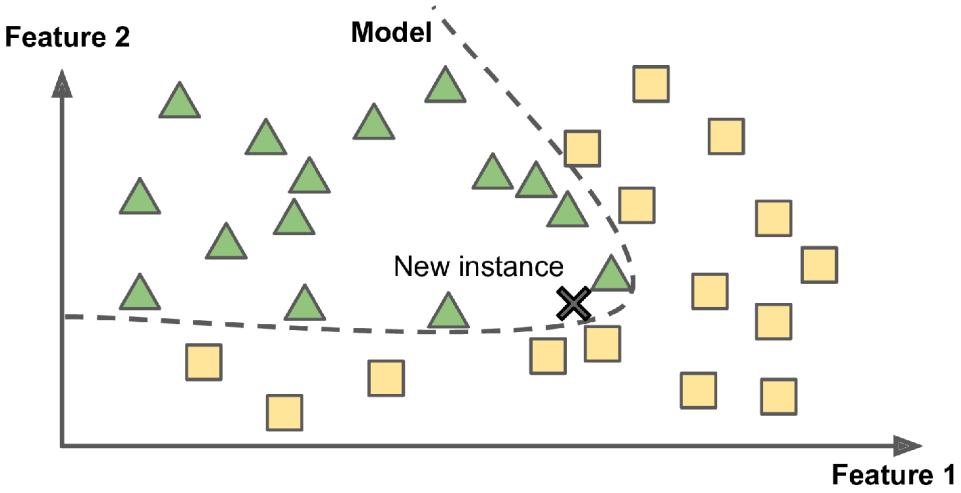

Figure 1.13: Model-based learning

Chapter 2

End-to-End Machine Learning Project

2.1 Working with Real Data

When you are learning about Machine Learning, it is best to experiment with realworld data, not artificial datasets. Here are a few places you can look to get data:

- Popular open data repositories
  - UC Irvine Machine Learning Repository
  - Kaggle datasets
  - Amazon's AWS datasets
- Meta portals (they list open data repositories)
  - Data Portals
  - OpenDataMonitor
  - Quandl
- Other pages listing many popular open data repositories
  - Wikipedia's list of Machine Learning datasets
  - Quora.com
  - The datasets subreddit

2.2 Look at the Big Picture

2.2.1 Frame the Problem

The first question to ask your boss is what exactly the business objective is. Building a model is probably not the end goal. How does the company expect to use and benefit from this model? Knowing the objective is important because it will determine how you frame the problem, which algorithms you will select, which performance measure you will use to evaluate your model, and how much effort you will spend tweaking it.

Your boss answers that your model's output (a prediction of a district's median housing price) will be fed to another Machine Learning system (see **??**), along with many other signals[1].

[1] A piece of information fed to a Machine Learning system is often called a signal, in reference to Claude Shannon's information theory, which he developed at Bell Labs to improve telecommunications. His theory: you want a high signal-to-noise ratio.

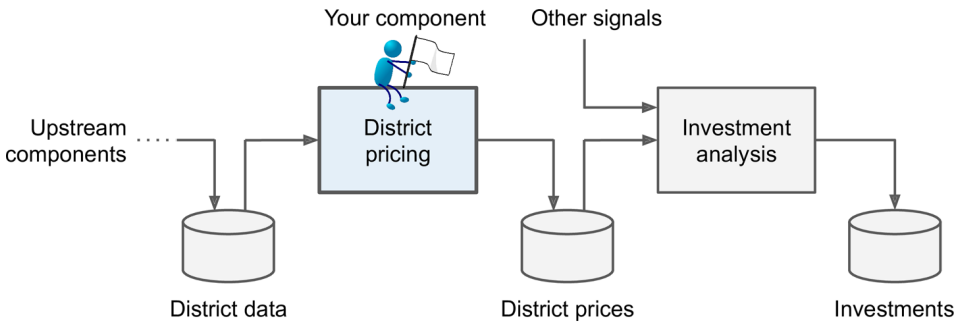

Figure 2.1: A Machine Learning pipeline for real estate investments

Pipelines

A sequence of data processing components is called a data pipeline. Pipelines are very common in Machine Learning systems, since there is a lot of data to manipulate and many data transformations to apply.
Components typically run asynchronously. Each component pulls in a large amount of data, processes it, and spits out the result in another data store. Then, some time later, the next component in the pipeline pulls this data and spits out its own output. Each component is fairly self-contained: the interface between components is simply the data store. This makes the system simple to grasp (with the help of a data flow graph), and different teams can focus on different components. Moreover, if a component breaks down, the downstream components can often continue to run normally (at least for a while) by just using the last output from the broken component. This makes the architecture quite robust.
On the other hand, a broken component can go unnoticed for some time if proper monitoring is not implemented. The data gets stale and the overall system's performance drops.

The next question to ask your boss is what the current solution looks like (if any). The current situation will often give you a reference for performance

Tips: If the data were huge, you could either split your batch learning work across multiple servers (using the MapReduce technique) or use an online learning technique.

2.2.2 Select a Performance Measure

Your next step is to select a performance measure. A typical performance measure for regression problems is the Root Mean Square Error (RMSE). It gives an idea of how much error the system typically makes in its predictions, with a higher weight for large errors. **??** shows the mathematical formula to compute the RMSE.

$$RMSE(\mathbf{X},h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})^2} \tag{2.1}$$

Notations

This equation introduces several very common Machine Learning notations that we will use throughout this book:

- m is the number of instances in the dataset you are measuring the RMSE on.
- $\mathbf{x}^{(i)}$ is a vector of all the feature values (excluding the label) of the i^{th} instance in the dataset, and $y^{(i)}$ is its

label (the desired output value for that instance). e.g.,

$$\mathbf{x}^{(i)} = \begin{bmatrix} -118.29 \\ 33.91 \\ 1,416 \\ 38,372 \end{bmatrix}$$

- $\mathbf{X}$ is a matrix containing all the feature values (excluding labels) of all instances in the dataset. There is one row per instance, and the i^{th} row is equal to the transpose of $\mathbf{x}^{(i)}$, noted $(\mathbf{x}^{(i)})^T$. e.g.,

$$\mathbf{x}^{(i)} = \begin{bmatrix} (\mathbf{x}^{(1)})^T \\ (\mathbf{x}^{(2)})^T \\ \vdots \\ (\mathbf{x}^{(m-1)})^T \\ (\mathbf{x}^{(m)})^T \end{bmatrix} = \begin{bmatrix} -118.29 & 33.91 & 1,416 & 38,372 \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

- h is your system's prediction function, also called a *hypothesis*. When your system is given an instance's feature vector $\mathbf{x}^{(i)}$, it outputs a predicted value $\hat{y}^{(i)} = h(\mathbf{x}^{(i)})$ for that instance.
- $RMSE(\mathbf{X}, h)$ is the cost function measured on the set of examples using your hypothesis h.

  We use lowercase italic font for scalar values and function names, lowercase bold font for vectors, and uppercase bold font for matrices.

Even though the RMSE is generally the preferred performance measure for regression tasks, in some contexts you may prefer to use another function. For example, **suppose that there are many outlier districts. In that case, you may consider using the mean absolute error** (MAE, also called the average absolute deviation; see **??**):

$$MAE(\mathbf{X}, h) = \frac{1}{m} \sum_{i=1}^{m} |h(x^{(i)}) - y^{(i)}| \tag{2.2}$$

Both the RMSE and the MAE are ways to measure the distance between two vectors: the vector of predictions and the vector of target values. Various distance measures, or *norms*, are possible:

- Computing the root of a sum of squares (RMSE) corresponds to the *Euclidean norm*: this is the notion of distance you are familiar with. It is also called the l_2 *norm*, noted $||\cdot||_2$ (or just$||\cdot||$).
- Computing the sum of absolutes (MAE) corresponds to the l_1 *norm*, noted $||\cdot||_1$. This is sometimes called the *Manhattan norm* because it measures the distance between two points in a city if you can only travel along orthogonal city blocks.
- More generally, the l_k norm of a vector v containing n elements is defined as $||v||_k = (|v_0|^k + |v_1|^k + \cdots + |v_n|^k)^{1/k}$. l_0 gives the number of nonzero elements in the vector, and l_∞ gives the maximum absolute value in the vector.
- **The higher the norm index, the more it focuses on large values and neglects small ones.** This is why the RMSE is more sensitive to outliers than the MAE. But when outliers are exponentially rare (like in a bell-shaped curve), the RMSE performs very well and is generally preferred.

2.2.3 Check the Assumptions

Lastly, it is good practice to list and verify the assumptions that have been made so far (by you or others); this can help you catch serious issues early on. For example, the district prices that your system outputs are going to be fed into a downstream Machine Learning system, and you assume that these prices are going to be used as such. But what if the

downstream system converts the prices into categories (e.g., “cheap,” “medium,” or “expensive”) and then uses those categories instead of the pri- ces themselves? In this case, getting the price perfectly right is not important at all; your system just needs to get the category right. If that’s so, then the problem should have been framed as a classification task, not a regression task. You don’t want to find this out after working on a regression system for months.

2.3 Get the Data

The full Jupyter notebook is available at https://github.com/JPL-JUNO/HOML.

2.3.1 Create the Workspace

You will need to have Python installed. It is probably already installed on your system. If not, you can get it at https://www.python.org/.

If you already have Jupyter running with all these modules installed, you can safely skip to **??**.

2.3.2 Download the Data

Having a function that downloads the data is useful in particular if the data changes regularly: you can write a small script that uses the function to fetch the latest data (or you can set up a scheduled job to do that automatically at regular intervals). Automating the process of fetching the data is also useful if you need to install the dataset on multiple machines.

2.3.3 Take a Quick Look at the Data Structure

Let’s take a look at the five rows using the DataFrame’s `sample(n=5)` method instead of `head(n=5)`.

The `info()` method is useful to get a quick description of the data, in particular the total number of rows, each attribute’s type, and the number of nonnull values. Notice that the `total_bedrooms` attribute has only 20,433 nonnull values, meaning that 207 districts are missing this feature. We will need to take care of this later.

You can find out what cate- gories exist and how many districts belong to each category by using the `value_counts()` method:

```
housing['ocean_proximity'].value_counts()
```

The `describe()` method shows a summary of the numerical attributes. When with many columns, you can use `transpose()` for best information.

```
# housing.describe()
housing.describe().transpose()
```