O'REILLY®

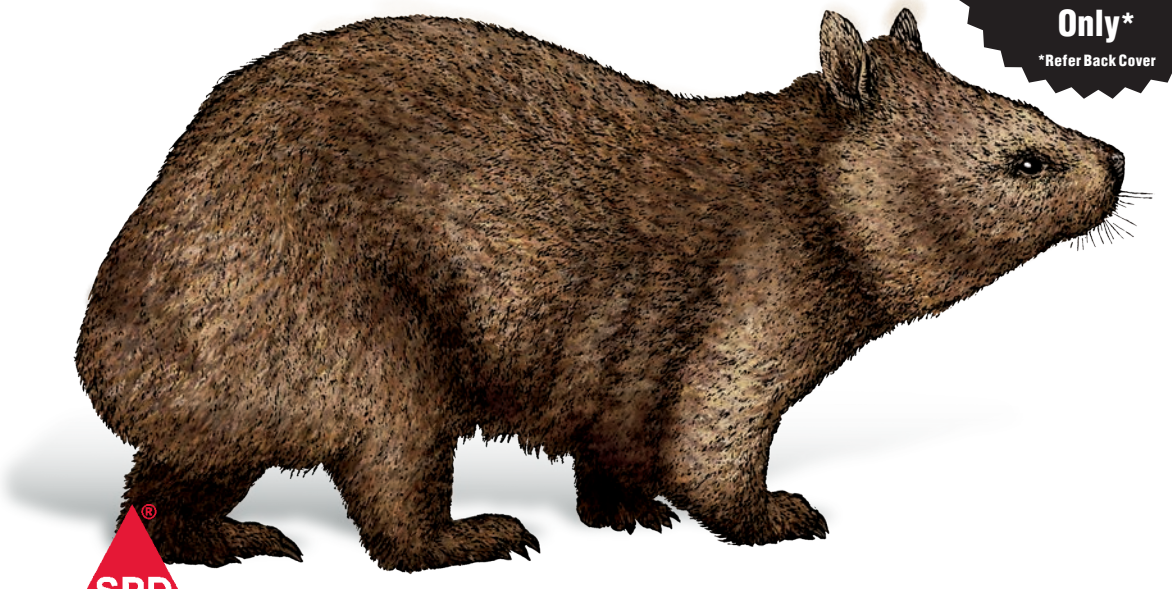
Hands-On Unsupervised Learning Using Python

How to Build Applied Machine Learning Solutions from Unlabeled Data

Grayscale Edition

For Sale in the Indian Subcontinent & Select Countries Only*

*Refer Back Cover



SPD®

Ankur A. Patel

Hands-On Unsupervised Learning Using Python

How to Build Applied Machine Learning Solutions from Unlabeled Data

Ankur A. Patel

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

SHROFF PUBLISHERS & DISTRIBUTORS PVT. LTD.

Mumbai Bangalore Kolkata New Delhi

Hands-On Unsupervised Learning Using Python

by Ankur A. Patel

 ISBN: 978-1-492-03564-0
Originally printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Developmental Editors: Michele Cronin
Acquisitions Editor: Jonathan Hassell
Production Editor: Katherine Tozer
Copyeditor: Jasmine Kwityn
Proofreader: Christina Edwards

Indexer: Judith McConville
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

Printing History:
February 2019: First Edition

Revision History for the First Edition 2019-02-21: First Release

See http://oreilly.com/catalog/errata.csp?isbn=9781492035640 for release details.

First Indian Reprint: April 2019

ISBN: 978-93-5213-812-8

Published by **Shroff Publishers & Distributors Pvt. Ltd.** B-103, Railway Commercial Complex, Sector 3, Sanpada (E), Navi Mumbai 400705 • TEL: (91 22) 4158 4158 • FAX: (91 22) 4158 4141 E-mail:spdorders@shroffpublishers.com•Web:www.shroffpublishers.com Printed at Jasmine Art Printers Pvt. Ltd., Mumbai.

Table of Contents

Part II. Unsupervised Learning Using Scikit-Learn

Part III. Unsupervised Learning Using TensorFlow and Keras

Preface

A Brief History of Machine Learning

Machine learning is a subfield of artificial intelligence (AI) in which computers learn from data—usually to improve their performance on some narrowly defined task—without being explicitly programmed. The term *machine learning* was coined as early as 1959 (by Arthur Samuel, a legend in the field of AI), but there were few major commercial successes in machine learning during the twenty-first century. Instead, the field remained a niche research area for academics at universities.

Early on (in the 1960s) many in the AI community were too optimistic about its future. Researchers at the time, such as Herbert Simon and Marvin Minsky, claimed that AI would reach human-level intelligence within a matter of decades:[1]

> Machines will be capable, within twenty years, of doing any work a man can do.
>
> —Herbert Simon, 1965

> From three to eight years, we will have a machine with the general intelligence of an average human being.
>
> —Marvin Minsky, 1970

Blinded by their optimism, researchers focused on so-called *strong AI* or *general artificial intelligence (AGI)* projects, attempting to build AI agents capable of problem solving, knowledge representation, learning and planning, natural language processing, perception, and motor control. This optimism helped attract significant funding into the nascent field from major players such as the Department of Defense, but the problems these researchers tackled were too ambitious and ultimately doomed to fail.

AI research rarely made the leap from academia to industry, and a series of so-called AI winters followed. In these AI winters (an analogy based on the nuclear winter dur-

1 Such views inspired Stanley Kubrick in 1968 to create the AI agent HAL 9000 in *2001: A Space Odyssey*.

ing this Cold War era), interest in and funding for AI dwindled. Occasionally, hype cycles around AI occurred but had very little staying power. By the early 1990s, interest in and funding for AI had hit a trough.

AI Is Back, but Why Now?

AI has re-emerged with a vengeance over the past two decades—first as a purely academic area of interest and now as a full-blown field attracting the brightest minds at both universities and corporations.

Three critical developments are behind this resurgence: breakthroughs in machine learning algorithms, the availability of lots of data, and superfast computers.

First, instead of focusing on overly ambitious strong AI projects, researchers turned their attention to narrowly defined subproblems of strong AI, also known as *weak AI* or *narrow AI*. This focus on improving solutions for narrowly defined tasks led to algorithmic breakthroughs, which paved the way for successful commercial applications. Many of these algorithms—often developed initially at universities or private research labs—were quickly open-sourced, speeding up the adoption of these technologies by industry.

Second, data capture became a focus for most organizations, and the costs of storing data fell dramatically driven by advances in digital data storage. Thanks to the internet, lots of data also became widely and publicly available at a scale never before seen.

Third, computers became increasingly powerful and available over the cloud, allowing AI researchers to easily and cheaply scale their IT infrastructure as required without making huge upfront investments in hardware.

The Emergence of Applied AI

These three forces have pushed AI from academia to industry, helping attract increasingly higher levels of interest and funding every year. AI is no longer just a theoretical area of interest but rather a full-blown applied field. Figure P-1 shows a chart from Google Trends, indicating the growth in interest in machine learning over the past five years.

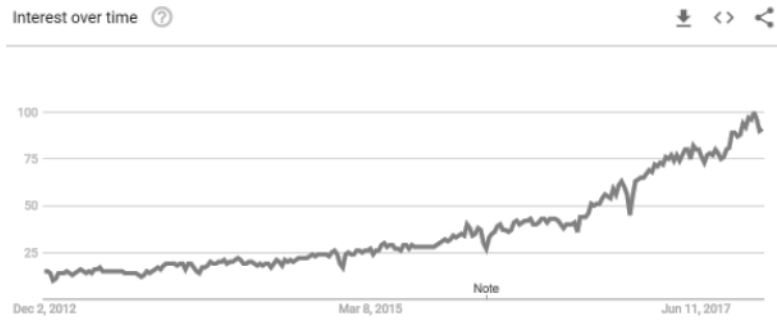


Figure P-1. Interest in machine learning over time

AI is now viewed as a breakthrough horizontal technology, akin to the advent of computers and smartphones, that will have a significant impact on every single industry over the next decade.[2]

Successful commercial applications involving machine learning include—but are certainly not limited to—optical character recognition, email spam filtering, image classification, computer vision, speech recognition, machine translation, group segmentation and clustering, generation of synthetic data, anomaly detection, cybercrime prevention, credit card fraud detection, internet fraud detection, time series prediction, natural language processing, board game and video game playing, document classification, recommender systems, search, robotics, online advertising, sentiment analysis, DNA sequencing, financial market analysis, information retrieval, question answering, and healthcare decision making.

Major Milestones in Applied AI over the Past 20 Years

The milestones presented here helped bring AI from a mostly academic topic of conversation then to a mainstream staple in technology today.

- 1997: Deep Blue, an AI bot that had been in development since the mid-1980s, beats world chess champion Garry Kasparov in a highly publicized chess event.
- 2004: DARPA introduces the DARPA Grand Challenge, an annually held autonomous driving challenge held in the desert. In 2005, Stanford takes the top prize. In 2007, Carnegie Mellon University performs this feat in an urban setting. In 2009, Google builds a self-driving car. By 2015, many major technology giants, including Tesla, Alphabet's Waymo, and Uber, have launched well-funded programs to build mainstream self-driving technology.

2 According to McKinsey Global Institute, over half of all the professional activities people are paid to do could be automated by 2055.

- 2006: Geoffrey Hinton of the University of Toronto introduces a fast learning algorithm to train neural networks with many layers, kicking off the deep learning revolution.
- 2006: Netflix launches the Netflix Prize competition, with a one million dollar purse, challenging teams to use machine learning to improve its recommendation system's accuracy by at least 10%. A team won the prize in 2009.
- 2007: AI achieves superhuman performance at checkers, solved by a team from the University of Alberta.
- 2010: ImageNet launches an annual contest—the ImageNet Large Scale Visual Recognition Challenge (ILSVRC)—in which teams use machine learning algorithms to correctly detect and classify objects in a large, well-curated image dataset. This draws significant attention from both academia and technology giants. The classification error rate falls from 25% in 2011 to just a few percent by 2015, backed by advances in deep convolutional neural networks. This leads to commercial applications of computer vision and object recognition.
- 2010: Microsoft launches Kinect for Xbox 360. Developed by the computer vision team at Microsoft Research, Kinect is capable of tracking human body movement and translating this into gameplay.
- 2010: Siri, one of the first mainstream digital voice assistants, is acquired by Apple and released as part of iPhone 4S in October 2011. Eventually, Siri is rolled out across all of Apple's products. Powered by convolutional neural networks and long short-term memory recurrent neural networks, Siri performs both speech recognition and natural language processing. Eventually, Amazon, Microsoft, and Google enter the race, releasing Alexa (2014), Cortana (2014), and Google Assistant (2016), respectively.
- 2011: IBM Watson, a question-answering AI agent developed by a team led by David Ferrucci, beats former *Jeopardy!* winners Brad Rutter and Ken Jennings. IBM Watson is now used across several industries, including healthcare and retail.
- 2012: Google Brain team, led by Andrew Ng and Jeff Dean, trains a neural network to recognize cats by watching unlabeled images taken from YouTube videos.
- 2013: Google wins DARPA's Robotics Challenge, involving trials in which semi-autonomous bots perform complex tasks in treacherous environments, such as driving a vehicle, walking across rubble, removing debris from a blocked entryway, opening a door, and climbing a ladder.
- 2014: Facebook publishes work on DeepFace, a neural network-based system that can identify faces with 97% accuracy. This is near human-level performance and is a more than 27% improvement over previous systems.

- 2015: AI goes mainstream, and is commonly featured in media outlets around the world.
- 2015: Google DeepMind's AlphaGo beats world-class professional Fan Hui at the game Go. In 2016, AlphaGo defeats Lee Sedol, and in 2017, AlphaGo defeats Ke Jie. In 2017, a new version called AlphaGo Zero defeats the previous AlphaGo version 100 to zero. AlphaGo Zero incorporates unsupervised learning techniques and masters Go just by playing itself.
- 2016: Google launches a major revamp to its language translation, Google Translate, replacing its existing phrase-based translation system with a deep learning-based neural machine translation system, reducing translation errors by up to 87% and approaching near human-level accuracy.
- 2017: Libratus, developed by Carnegie Mellon, wins at head-to-head no-limit Texas Hold'em.
- 2017: OpenAI-trained bot beats professional gamer at Dota 2 tournament.

From Narrow AI to AGI

Of course, these successes in applying AI to narrowly defined problems are just a starting point. There is a growing belief in the AI community that—by combining several weak AI systems—we can develop strong AI. This strong AI or AGI agent will be capable of human-level performance at many broadly defined tasks.

Soon after AI achieves human-level performance, some researchers predict this strong AI will surpass human intelligence and reach so-called *superintelligence*. Estimates for attaining such superintelligence range from as little as 15 years to as many as 100 years from now, but most researchers believe AI will advance enough to achieve this in a few generations. Is this inflated hype once again (like what we saw in previous AI cycles), or is it different this time around?

Only time will tell.

Objective and Approach

Most of the successful commercial applications to date—in areas such as computer vision, speech recognition, machine translation, and natural language processing—have involved supervised learning, taking advantage of labeled datasets. However, most of the world's data is *unlabeled*.

In this book, we will cover the field of *unsupervised learning* (which is a branch of machine learning used to find hidden patterns) and learn the underlying structure in unlabeled data. According to many industry experts, such as Yann LeCun, the Director of AI Research at Facebook and a professor at NYU, unsupervised learning is the

next frontier in AI and may hold the key to AGI. For this and many other reasons, unsupervised learning is one of the trendiest topics in AI today.

The book's goal is to outline the concepts and tools required for you to develop the intuition necessary for applying this technology to everyday problems that you work on. In other words, this is an applied book, one that will allow you to build real-world systems. We will also explore how to efficiently label unlabeled datasets to turn unsupervised learning problems into semisupervised ones.

The book will use a hands-on approach, introducing some theory but focusing mostly on applying unsupervised learning techniques to solving real-world problems. The datasets and code are available online as Jupyter notebooks on GitHub (*http://bit.ly/2Gd4v7e*).

Armed with the conceptual understanding and hands-on experience you'll gain from this book, you will be able to apply unsupervised learning to large, unlabeled datasets to uncover hidden patterns, obtain deeper business insight, detect anomalies, cluster groups based on similarity, perform automatic feature engineering and selection, generate synthetic datasets, and more.

Prerequisites

This book assumes that you have some Python programming experience, including familiarity with NumPy and Pandas.

For more on Python, visit the official Python website (*https://www.python.org/*). For more on Jupyter Notebook, visit the official Jupyter website (*http://jupyter.org/index.html*). For a refresher on college-level calculus, linear algebra, probability, and statistics, read Part I of the *Deep Learning* textbook (*http://www.deeplearningbook.org/*) by Ian Goodfellow and Yoshua Bengio. For a refresher on machine learning, read *The Elements of Statistical Learning* (*https://stanford.io/2Tju4al*).

Roadmap

The book is organized into four parts, covering the following topics:

Part I, Fundamentals of Unsupervised Learning
: Differences between supervised and unsupervised learning, an overview of popular supervised and unsupervised algorithms, and an end-to-end machine learning project

Part II, Unsupervised Learning Using Scikit-Learn
: Dimensionality reduction, anomaly detection, and clustering and group segmentation

For more information on the concepts discussed in Parts I and II, refer to the Scikit-learn documentation (*https://scikit-learn.org/stable/modules/classes.html*).

Part III, Unsupervised Learning Using TensorFlow and Keras
: Representation learning and automatic feature extraction, autoencoders, and semisupervised learning

Part IV, Deep Unsupervised Learning Using TensorFlow and Keras
: Restricted Boltzmann machines, deep belief networks, and generative adversarial networks

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

This element signifies a tip or suggestion.

This element signifies a general note.

This element indicates a warning or caution.

Using Code Examples

Supplemental material (code examples, etc.) is available for download on GitHub (*http://bit.ly/2Gd4v7e*).

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Hands-On Unsupervised Learning Using Python* by Ankur A. Patel (O'Reilly). Copyright 2019 Ankur A. Patel, 978-1-492-03564-0."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

O'Reilly Online Learning

For almost 40 years, *O'Reilly Media* has provided technology and business training, knowledge, and insight to help companies succeed.

Our unique network of experts and innovators share their knowledge and expertise through books, articles, conferences, and our online learning platform. O'Reilly's online learning platform gives you on-demand access to live training courses, in-depth learning paths, interactive coding environments, and a vast collection of text and video from O'Reilly and 200+ other publishers. For more information, please visit *http://oreilly.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *http://bit.ly/unsupervised-learning*.

To comment or ask technical questions about this book, send email to *bookquestions@oreilly.com*.

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

PART I

Fundamentals of Unsupervised Learning

To start, let's explore the current machine learning ecosystem and where unsupervised learning fits in. We will also build a machine learning project from scratch to cover basics such as setting up the programming environment, acquiring and preparing data, exploring data, selecting machine learning algorithms and cost functions, and evaluating the results.

CHAPTER 1

Unsupervised Learning in the Machine Learning Ecosystem

> Most of human and animal learning is unsupervised learning. If intelligence was a cake, unsupervised learning would be the cake, supervised learning would be the icing on the cake, and reinforcement learning would be the cherry on the cake. We know how to make the icing and the cherry, but we don't know how to make the cake. We need to solve the unsupervised learning problem before we can even think of getting to true AI.
>
> —Yann LeCun

In this chapter, we will explore the difference between a rules-based system and machine learning, the difference between supervised learning and unsupervised learning, and the relative strengths and weaknesses of each.

We will also cover many popular supervised learning algorithms and unsupervised learning algorithms and briefly examine how semisupervised learning and reinforcement learning fit into the mix.

Basic Machine Learning Terminology

Before we delve into the different types of machine learning, let's take a look at a simple and commonly used machine learning example to help make the concepts we introduce tangible: the email spam filter. We need to build a simple program that takes in emails and correctly classifies them as either "spam" or "not spam." This is a straightforward classification problem.

Here's a bit of machine learning terminology as a refresher: the *input variables* into this problem are the text of the emails. These input variables are also known as *features* or *predictors* or *independent variables*. The *output variable*—what we are trying

to predict—is the *label* "spam" or "not spam." This is also known as the *target variable*, *dependent variable*, or *response variable* (or *class* since this is a classification problem).

The set of examples the AI trains on is known as the *training set*, and each individual example is called a training *instance* or *sample*. During the training, the AI is attempting to minimize its *cost function* or *error rate*, or framed more positively, to maximize its *value function*—in this case, the ratio of correctly classified emails. The AI actively optimizes for a minimal error rate during training. Its error rate is calculated by comparing the AI's predicted label with the true label.

However, what we care about most is how well the AI generalizes its training to never-before-seen emails. This will be the true test for the AI: can it correctly classify emails that it has never seen before using what it has learned by training on the examples in the training set? This *generalization error* or *out-of-sample error* is the main thing we use to evaluate machine learning solutions.

This set of never-before-seen examples is known as the *test set* or *holdout set* (because the data is held out from the training). If we choose to have multiple holdout sets (perhaps to gauge our generalization error as we train, which is advisable), we may have intermediate holdout sets that we use to evaluate our progress before the final test set; these intermediate holdout sets are called *validation sets*.

To put all of this together, the AI trains on the training data (*experience*) to improve its error rate (*performance*) in flagging spam (*task*), and the ultimate success criterion is how well its experience generalizes to new, never-before-seen data (*generalization error*).

Rules-Based vs. Machine Learning

Using a rules-based approach, we can design a spam filter with explicit rules to catch spam such as flag emails with "u" instead of "you," "4" instead of "for," "BUY NOW," etc. But this system would be difficult to maintain over time as bad guys change their spam behavior to evade the rules. If we used a rules-based system, we would have to frequently adjust the rules manually just to stay up-to-date. Also, it would be very expensive to set up—think of all the rules we would need to create to make this a well-functioning system.

Instead of a rules-based approach, we can use machine learning to train on the email data and automatically engineer rules to correctly flag malicious email as spam. This machine learning-based system could be automatically adjusted over time as well. This system would be much cheaper to train and maintain.

In this simple email problem, it may be possible for us to handcraft rules, but, for many problems, handcrafting rules is not feasible at all. For example, consider designing a self-driving car—imagine drafting rules for how the car should behave in

each and every single instance it ever encounters. This is an intractable problem unless the car can learn and adapt on its own based on its experience.

We could also use machine learning systems as an exploration or data discovery tool to gain deeper insight into the problem we are trying to solve. For example, in the email spam filter example, we can learn which words or phrases are most predictive of spam and recognize newly emerging malicious spam patterns.

Supervised vs. Unsupervised

The field of machine learning has two major branches—*supervised learning* and *unsupervised learning*—and plenty of sub-branches that bridge the two.

In supervised learning, the AI agent has access to labels, which it can use to improve its performance on some task. In the email spam filter problem, we have a dataset of emails with all the text within each and every email. We also know which of these emails are spam or not (the so-called *labels*). These labels are very valuable in helping the supervised learning AI separate the spam emails from the rest.

In unsupervised learning, labels are not available. Therefore, the task of the AI agent is not well-defined, and performance cannot be so clearly measured. Consider the email spam filter problem—this time without labels. Now, the AI agent will attempt to understand the underlying structure of emails, separating the database of emails into different groups such that emails within a group are similar to each other but different from emails in other groups.

This unsupervised learning problem is less clearly defined than the supervised learning problem and harder for the AI agent to solve. But, if handled well, the solution is more powerful.

Here's why: the unsupervised learning AI may find several groups that it later tags as being "spam"—but the AI may also find groups that it later tags as being "important" or categorize as "family," "professional," "news," "shopping," etc. In other words, because the problem does not have a strictly defined task, the AI agent may find interesting patterns above and beyond what we initially were looking for.

Moreover, this unsupervised system is better than the supervised system at finding new patterns in future data, making the unsupervised solution more nimble on a go-forward basis. This is the power of unsupervised learning.

The Strengths and Weaknesses of Supervised Learning

Supervised learning excels at optimizing performance in well-defined tasks with plenty of labels. For example, consider a very large dataset of images of objects, where each image is labeled. If the dataset is sufficiently large enough and we train using the right machine learning algorithms (i.e., convolutional neural networks) and with

powerful enough computers, we can build a very good supervised learning-based image classification system.

As the supervised learning AI trains on the data, it will be able to measure its performance (via a cost function) by comparing its predicted image label with the true image label that we have on file. The AI will explicitly try to minimize this cost function such that its error on never-before-seen images (from a holdout set) is as low as possible.

This is why labels are so powerful—they help guide the AI agent by providing it with an error measure. The AI uses the error measure to improve its performance over time. Without such labels, the AI does not know how successful it is (or isn't) in correctly classifying images.

However, the costs of manually labeling an image dataset are high. And, even the best curated image datasets have only thousands of labels. This is a problem because supervised learning systems will be very good at classifying images of objects for which it has labels but poor at classifying images of objects for which it has no labels.

As powerful as supervised learning systems are, they are also limited at generalizing knowledge beyond the labeled items they have trained on. Since the majority of the world's data is unlabeled, with supervised learning, the ability of AI to expand its performance to never-before-seen instances is quite limited.

In other words, supervised learning is great at solving narrow AI problems but not so good at solving more ambitious, less clearly defined problems of the strong AI type.

The Strengths and Weaknesses of Unsupervised Learning

Supervised learning will trounce unsupervised learning at narrowly defined tasks for which we have well-defined patterns that do not change much over time and sufficiently large, readily available labeled datasets.

However, for problems where patterns are unknown or constantly changing or for which we do not have sufficiently large labeled datasets, unsupervised learning truly shines.

Instead of being guided by labels, unsupervised learning works by learning the underlying structure of the data it has trained on. It does this by trying to represent the data it trains on with a set of parameters that is significantly smaller than the number of examples available in the dataset. By performing this representation learning, unsupervised learning is able to identify distinct patterns in the dataset.

In the image dataset example (this time without labels), the unsupervised learning AI may be able to identify and group images based on how similar they are to each other and how different they are from the rest. For example, all the images that look like

chairs will be grouped together, all the images that look like dogs will be grouped together, etc.

Of course, the unsupervised learning AI itself cannot label these groups as "chairs" or "dogs" but now that similar images are grouped together, humans have a much simpler labeling task. Instead of labeling millions of images by hand, humans can manually label all the distinct groups, and the labels will apply to all the members within each group.

After the initial training, if the unsupervised learning AI finds images that do not belong to any of the labeled groups, the AI will create separate groups for the unclassified images, triggering a human to label the new, yet-to-be-labeled groups of images.

Unsupervised learning makes previously intractable problems more solvable and is much more nimble at finding hidden patterns both in the historical data that is available for training and in future data. Moreover, we now have an AI approach for the huge troves of unlabeled data that exist in the world.

Even though unsupervised learning is less adept than supervised learning at solving specific, narrowly defined problems, it is better at tackling more open-ended problems of the strong AI type and at generalizing this knowledge.

Just as importantly, unsupervised learning can address many of the common problems data scientists encounter when building machine learning solutions.

Using Unsupervised Learning to Improve Machine Learning Solutions

Recent successes in machine learning have been driven by the availability of lots of data, advances in computer hardware and cloud-based resources, and breakthroughs in machine learning algorithms. But these successes have been in mostly narrow AI problems such as image classification, computer vision, speech recognition, natural language processing, and machine translation.

To solve more ambitious AI problems, we need to unlock the value of unsupervised learning. Let's explore the most common challenges data scientists face when building solutions and how unsupervised learning can help.

Insufficient labeled data

> I think AI is akin to building a rocket ship. You need a huge engine and a lot of fuel. If you have a large engine and a tiny amount of fuel, you won't make it to orbit. If you have a tiny engine and a ton of fuel, you can't even lift off. To build a rocket you need a huge engine and a lot of fuel.
>
> —Andrew Ng

If machine learning were a rocket ship, data would be the fuel—without lots and lots of data, the rocket ship cannot fly. But not all data is created equal. To use supervised algorithms, we need lots of labeled data, which is hard and costly to generate.[1]

With unsupervised learning, we can automatically label unlabeled examples. Here is how it would work: we would cluster all the examples and then apply the labels from labeled examples to the unlabeled ones within the same cluster. Unlabeled examples would receive the label of the labeled ones they are most similar to. We will explore clustering in Chapter 5.

Overfitting

If the machine learning algorithm learns an overly complex function based on the training data, it may perform very poorly on never-before-seen instances from hold-out sets such as the validation set or test set. In this case, the algorithm has overfit the training data—by extracting too much from the noise in the data—and has very poor generalization error. In other words, the algorithm is memorizing the training data rather than learning how to generalize knowledge based off of it.[2]

To address this, we can introduce unsupervised learning as a *regularizer*. *Regularization* is a process used to reduce the complexity of a machine learning algorithm, helping it capture the signal in the data without adjusting too much to the noise. Unsupervised pretraining is one such form of regularization. Instead of feeding the original input data directly into a supervised learning algorithm, we can feed a new representation of the original input data that we generate.

This new representation captures the essence of the original data—the true underlying structure—while losing some of the less representative noise along the way. When we feed this new representation into the supervised learning algorithm, it has less noise to wade through and captures more of the signal, improving its generalization error. We will explore feature extraction in Chapter 7.

Curse of dimensionality

Even with the advances in computational power, big data is hard for machine learning algorithms to manage. In general, adding more instances is not too problematic because we can parallelize operations using modern map-reduce solutions such as Spark. However, the more features we have, the more difficult training becomes.

1 There are startups such as Figure Eight that explicitly provide this *human in the loop* service.

2 Underfitting is another problem that may occur in building machine learning applications, but this is easier to solve. Underfitting occurs because the model is too simple—the algorithm cannot build a complex enough function approximation to make good decisions for the task at hand. To solve this, we can allow the algorithm to grow in size (have more parameters, perform more training iterations, etc.) or apply a more complicated machine learning algorithm.

In a very high-dimensional space, supervised algorithms need to learn how to separate points and build a function approximation to make good decisions. When the features are very numerous, this search becomes very expensive, both from a time and compute perspective. In some cases, it may be impossible to find a good solution fast enough.

This problem is known as the *curse of dimensionality*, and unsupervised learning is well suited to help manage this. With dimensionality reduction, we can find the most salient features in the original feature set, reduce the number of dimensions to a more manageable number while losing very little important information in the process, and then apply supervised algorithms to more efficiently perform the search for a good function approximation. We will cover dimensionality reduction in Chapter 3.

Feature engineering

Feature engineering is one of the most vital tasks data scientists perform. Without the right features, the machine learning algorithm will not be able to separate points in space well enough to make good decisions on never-before-seen examples. However, feature engineering is typically very labor-intensive; it requires humans to creatively hand-engineer the right types of features. Instead, we can use representation learning from unsupervised learning algorithms to automatically learn the right types of feature representations to help solve the task at hand. We will explore automatic feature extraction in Chapter 7.

Outliers

The quality of data is also very important. If machine learning algorithms train on rare, distortive outliers, their generalization error will be lower than if they ignored or addressed the outliers separately. With unsupervised learning, we can perform outlier detection using dimensionality reduction and create a solution specifically for the outliers and, separately, a solution for the normal data. We will build an anomaly detection system in Chapter 4.

Data drift

Machine learning models also need to be aware of drift in the data. If the data the model is making predictions on differs statistically from the data the model trained on, the model may need to retrain on data that is more representative of the current data. If the model does not retrain or does not recognize the drift, the model's prediction quality on current data will suffer.

By building probability distributions using unsupervised learning, we can assess how different the current data is from the training set data—if the two are different enough, we can automatically trigger a retraining. We will explore how to build these types of data discriminators in Chapter 12.

A Closer Look at Supervised Algorithms

Before we delve into unsupervised learning systems, let's take a look at supervised learning algorithms and how they work. This will help frame where unsupervised learning fits within the machine learning ecosystem.

In supervised learning, there are two major types of problems: *classification* and *regression*. In classification, the AI must correctly classify items into one of two or more classes. If there are just two classes, the problem is called *binary classification*. If there are three or more classes, the problem is classed *multiclass classification*.

Classification problems are also known as *discrete* prediction problems because each class is a discrete group. Classification problems also may be referred to as *qualitative* or *categorical* problems.

In regression, the AI must predict a *continuous* variable rather than a discrete one. Regression problems also may be referred to as *quantitative* problems.

Supervised machine learning algorithms span the gamut, from very simple to very complex, but they are all aimed at minimizing some cost function or error rate (or maximizing a value function) that is associated with the labels we have for the dataset.

As mentioned before, what we care about most is how well the machine learning solution generalizes to never-before-seen cases. The choice of the supervised learning algorithm is very important at minimizing this generalization error.

To achieve the lowest possible generalization error, the complexity of the algorithmic model should match the complexity of the true function underlying the data. We do not know what this true function really is. If we did, we would not need to use machine learning to create a model—we would just solve the function to find the right answer. But since we do not know what this true function is, we choose a machine learning algorithm to test hypotheses and find the model that best approximates this true function (i.e., has the lowest possible generalization error).

If what the algorithm models is less complex than the true function, we have *underfit* the data. In this case, we could improve the generalization error by choosing an algorithm that can model a more complex function. However, if the algorithm designs an overly complex model, we have *overfit* the training data and will have poor performance on never-before-seen cases, increasing our generalization error.

In other words, choosing more complex algorithms over simpler ones is not always the right choice—sometimes simpler is better. Each algorithm comes with its set of strengths, weaknesses, and assumptions, and knowing what to use when given the data you have and the problem you are trying to solve is very important to mastering machine learning.

In the rest of this chapter, we will describe some of the most common supervised algorithms (including some real-world applications) before doing the same for unsupervised algorithms.[3]

Linear Methods

The most basic supervised learning algorithms model a simple linear relationship between the input features and the output variable that we wish to predict.

Linear regression

The simplest of all the algorithms is *linear regression*, which uses a model that assumes a linear relationship between the input variables (x) and the single output variable (y). If the true relationship between the inputs and the output is linear and the input variables are not highly correlated (a situation known as *collinearity*), linear regression may be an appropriate choice. If the true relationship is more complex or nonlinear, linear regression will underfit the data.[4]

Because it is so simple, interpreting the relationship modeled by the algorithm is also very straightforward. *Interpretability* is a very important consideration for applied machine learning because solutions need to be understood and enacted by both technical and nontechnical people in industry. Without interpretability, the solutions become inscrutable black boxes.

Strengths
: Linear regression is simple, intrepretable, and hard to overfit because it cannot model overly complex relationships. It is an excellent choice when the underlying relationship between the input and output variables is linear.

Weaknesses
: Linear regression will underfit the data when the relationship between the input and output variables is nonlinear.

Applications
: Since the true underlying relationship between human weight and human height is linear, linear regression is great for predicting weight using height as the input variable or, vice versa, for predicting height using weight as the input variable.

3 This list is by no means exhaustive but does include the most commonly used machine learning algorithms.

4 There may be other potential issues that might make linear regression a poor choice, including outliers, correlation of error terms, and nonconstant variance of error terms.

Logistic regression

The simplest classification algorithm is *logistic regression*, which is also a linear method but the predictions are transformed using the logistic function. The outputs of this transformation are *class probabilities*—in other words, the probabilities that the instance belongs to the various classes, where the sum of the probabilities for each instance adds up to one. Each instance is then assigned to the class for which it has the highest probability of belonging in.

Strengths
: Like linear regression, logistic regression is simple and interpretable. When the classes we are trying to predict are nonoverlapping and linearly separable, logistic regression is an excellent choice.

Weaknesses
: When classes are not linearly separable, logistic regression will fail.

Applications
: When classes are mostly nonoverlapping—for example, the heights of young children versus the heights of adults—logistic regression will work well.

Neighborhood-Based Methods

Another group of very simple algorithms are neighborhood-based methods. Neighborhood-based methods are *lazy learners* since they learn how to label new points based on the proximity of the new points to existing labeled points. Unlike linear regression or logistic regression, neighborhood-based models do not learn a set model to predict labels for new points; rather, these models predict labels for new points based purely on distance of new points to preexisting labeled points. Lazy learning is also referred to as *instance-based learning* or *nonparametric methods*.

k-nearest neighbors

The most common neighborhood-based method is *k-nearest neighbors (KNN)*. To label each new point, KNN looks at a k number (where k is an integer value) of nearest labeled points and has these already labeled neighbors vote on how to label the new point. By default, KNN uses Euclidean distance to measure what is closest.

The choice of k is very important. If k is set to a very low value, KNN becomes very flexible, drawing highly nuanced boundaries and potentially overfitting the data. If k is set to a very high value, KNN becomes inflexible, drawing a too rigid boundary and potentially underfitting the data.

Strengths
: Unlike linear methods, KNN is highly flexible and adept at learning more complex, nonlinear relationships. Yet, KNN remains simple and interpretable.

Weaknesses

KNN does poorly when the number of observations and features grow. KNN becomes computationally inefficient in this highly populated, high-dimensional space since it needs to calculate distances from the new point to many nearby labeled points in order to predict labels. It cannot rely on an efficient model with a reduced number of parameters to make the necessary prediction. Also, KNN is very sensitive to the choice of k. When k is set too low, KNN can overfit, and when k is set too high, KNN can underfit.

Applications

KNN is regularly used in recommender systems, such as those used to predict taste in movies (Netflix), music (Spotify), friends (Facebook), photos (Instagram), search (Google), and shopping (Amazon). For example, KNN can help predict what a user will like given what similar users like (known as *collaborative filtering*) or what the user has liked in the past (known as *content-based filtering*).

Tree-Based Methods

Instead of using a linear method, we can have the AI build a *decision tree* where all the instances are *segmented* or *stratified* into many regions, guided by the labels we have. Once this segmentation is complete, each region corresponds to a particular class of label (for classification problems) or a range of predicted values (for regression problems). This process is similar to having the AI build rules automatically with the explicit goal of making better decisions or predictions.

Single decision tree

The simplest tree-based method is a *single decision tree*, in which the AI goes once through the training data, creates rules for segmenting the data guided by the labels, and uses this tree to make predictions on the never-before-seen validation or test set. However, a single decision tree is usually poor at generalizing what it has learned during training to never-before-seen cases because it usually overfits the training data during its one and only training iteration.

Bagging

To improve the single decision tree, we can introduce *bootstrap aggregation* (more commonly known as *bagging*), in which we take *multiple random samples of instances* from the training data, create a decision tree for each sample, and then predict the output for each instance by averaging the predictions of each of these trees. By using *randomization* of samples and averaging results from multiple trees—an approach that is also known as the *ensemble method*—bagging will address some of the overfitting that results from a single decision tree.

Random forests

We can improve overfitting further by sampling not only the instances but also the predictors. With *random forests*, we take multiple random samples of instances from the training data like we do in bagging, but, for each split in each decision tree, we make the split based not on all the predictors but rather a *random sample of the predictors*. The number of predictors we consider for each split is usually the square root of the total number of predictors.

By sampling the predictors in this way, the random forests algorithm creates trees that are even less correlated with each other (compared to the trees in bagging), reducing overfitting and improving the generalization error.

Boosting

Another approach, known as *boosting*, is used to create multiple trees like in bagging but to *build the trees sequentially*, using what the AI learned from the previous tree to improve results on the subsequent tree. Each tree is kept pretty shallow, with only a few decision splits, and the learning occurs slowly, tree by tree. Of all the tree-based methods, *gradient boosting machines* are among the best-performing and are commonly used to win machine learning competitions.[5]

Strengths
: Tree-based methods are among the best-performing supervised-learning algorithms for prediction problems. These methods are able to capture complex relationships in the data by learning many simple rules, one rule at a time. They are also capable of handling missing data and categorical features.

Weaknesses
: Tree-based methods are difficult to interpret, especially if many rules are needed to make a good prediction. Performance also becomes an issue as the number of features increase.

Applications
: Gradient boosting and random forests are excellent for prediction problems.

Support Vector Machines

Instead of building trees to separate data, we can use algorithms to create hyperplanes in space that separate the data, guided by the labels that we have. The approach is known as *support vector machines (SVMs)*. SVMs allow some violations to this separation—not all the points within an area in hyperspace need to have the same label—

5 For more on gradient boosting in machine learning competitions, consult Ben Gorman's blog post (*http://bit.ly/2S1C8Qy*).

but the distance between boundary-defining points of a certain label and the boundary-defining points of another label should be maximized as much as possible. Also, the boundaries do not have to be linear—we can use nonlinear kernels to more flexibly separate the data.

Neural Networks

We can learn representations of the data using neural networks, which are composed of an input layer, several hidden layers, and an output layer.[6] The input layer uses the features, and the output layer tries to match the response variable. The hidden layers are a nested hierarchy of concepts—each layer (or concept) is trying to understand how the previous layer relates to the output layer.

Using this hierarchy of concepts, the neural network is able to learn complicated concepts by building them out of simpler ones. Neural networks are one of the most powerful approaches to function approximation but are prone to overfitting and are hard to interpret, shortcomings that we will explore in greater detail later in the book.

A Closer Look at Unsupervised Algorithms

We will now turn our attention to problems where we do not have labels. Instead of trying to make predictions, unsupervised learning algorithms will try to learn the underlying structure of the data.

Dimensionality Reduction

One family of algorithms—known as *dimensionality reduction algorithms*—projects the original high-dimensional input data to a low-dimensional space, filtering out the not-so-relevant features and keeping as much of the interesting ones as possible. Dimensionality reduction allows unsupervised learning AI to more effectively identify patterns and more efficiently solve large-scale, computationally expensive problems (often involving images, video, speech, and text).

Linear projection

There are two major branches of dimensionality—linear projection and nonlinear dimensionality reduction. We will start with linear projection first.

Principal component analysis (PCA). One approach to learning the underlying structure of data is to identify which features out of the full set of features are most important in explaining the variability among the instances in the data. Not all features are equal

6 For more on neutral networks, check out *Deep Learning* (*http://www.deeplearningbook.org/*) by Ian Goodfellow, Yoshua Bengio, and Aaron Courville (MIT Press).

—for some features, the values in the dataset do not vary much, and these features are less useful in explaining the dataset. For other features, the values might vary considerably—these features are worth exploring in greater detail since they will be better at helping the model we design separate the data.

In *PCA*, the algorithm finds a low-dimensional representation of the data while retaining as much of the variation as possible. The number of dimensions we are left with is considerably smaller than the number of dimensions of the full dataset (i.e., the number of total features). We lose some of the variance by moving to this low-dimensional space, but the underlying structure of the data is easier to identify, allowing us to perform tasks like clustering more efficiently.

There are several variants of PCA, which we will explore later in the book. These include mini-batch variants such as *incremental PCA*, nonlinear variants such as *kernel PCA*, and sparse variants such as *sparse PCA*.

Singular value decomposition (SVD). Another approach to learning the underlying structure of the data is to reduce the rank of the original matrix of features to a smaller rank such that the original matrix can be recreated using a linear combination of some of the vectors in the smaller rank matrix. This is known as *SVD*. To generate the smaller rank matrix, SVD keeps the vectors of the original matrix that have the most information (i.e., the highest singular value). The smaller rank matrix captures the most important elements of the original feature space.

Random projection. A similar dimensionality reduction algorithm involves projecting points from a high-dimensional space to a space of much lower dimensions in such a way that the scale of distances between the points is preserved. We can use either a *random Gaussian matrix* or a *random sparse matrix* to accomplish this.

Manifold learning

Both PCA and random projection rely on projecting the data linearly from a high-dimensional space to a low-dimensional space. Instead of a linear projection, it may be better to perform a nonlinear transformation of the data—this is known as *manifold learning* or *nonlinear dimensionality reduction*.

Isomap. *Isomap* is one type of manifold learning approach. This algorithm learns the intrinsic geometry of the data manifold by estimating the *geodesic* or *curved distance* between each point and its neighbors rather than the Euclidean distance. Isomap uses this to then embed the original high-dimensional space to a low-dimensional one.

t-distributed stochastic neighbor embedding (t-SNE). Another nonlinear dimensionality reduction—known as *t-SNE*—embeds high-dimensional data into a space of just two or three dimensions, allowing the transformed data to be visualized. In this two- or

three-dimensional space, similar instances are modeled closer together and dissimilar instances are modeled further away.

Dictionary learning. An approach known as *dictionary learning* involves learning the sparse representation of the underlying data. These representative elements are simple, binary vectors (zeros and ones), and each instance in the dataset can be reconstructed as a weighted sum of the representative elements. The matrix (known as the *dictionary*) that this unsupervised learning generates is mostly populated by zeros with only a few nonzero weights.

By creating such a dictionary, this algorithm is able to efficiently identify the most salient representative elements of the original feature space—these are the ones that have the most nonzero weights. The representative elements that are less important will have few nonzero weights. As with PCA, dictionary learning is excellent for learning the underlying structure of the data, which will be helpful in separating the data and in identifying interesting patterns.

Independent component analysis

One common problem with unlabeled data is that there are many independent signals embedded together into the features we are given. Using *independent component analysis (ICA)*, we can separate these blended signals into their individual components. After the separation is complete, we can reconstruct any of the original features by adding together some combination of the individual components we generate. ICA is commonly used in signal processing tasks (for example, to identify the individual voices in an audio clip of a busy coffeehouse).

Latent Dirichlet allocation

Unsupervised learning can also explain a dataset by learning why some parts of the dataset are similar to each other. This requires learning unobserved elements within the dataset—an approach known as *latent Dirichlet allocation (LDA)*. For example, consider a document of text with many, many words. These words within a document are not purely random; rather, they exhibit some structure.

This structure can be modeled as unobserved elements known as topics. After training, LDA is able to explain a given document with a small set of topics, where for each topic there is a small set of frequently used words. This is the hidden structure the LDA is able to capture, helping us better explain a previously unstructured corpus of text.

Dimensionality reduction reduces the original set of features to a smaller set of just the most important features. From here, we can run other unsupervised learning algorithms on this smaller set of features to find interesting patterns in the data (see the next section on clustering), or, if we have labels, we can speed up the training cycle of supervised learning algorithms by feeding in this smaller matrix of features instead of using the original feature matrix.

Clustering

Once we have reduced the set of original features to a smaller, more manageable set, we can find interesting patterns by grouping similar instances of data together. This is known as clustering and can be accomplished with a variety of unsupervised learning algorithms and be used for real-world applications such as market segmentation.

k-means

To cluster well, we need to identify distinct groups such that the instances within a group are similar to each other but different from instances in other groups. One such algorithm is *k-means clustering*. With this algorithm, we specify the number of desired clusters *k*, and the algorithm will assign each instance to exactly one of these *k* clusters. It optimizes the grouping by minimizing the *within-cluster variation* (also known as *inertia*) such that the sum of the within-cluster variations across all *k* clusters is as small as possible.

To speed up this clustering process, *k*-means randomly assigns each observation to one of the *k* clusters and then begins to reassign these observations to minimize the Euclidean distance between each observation and its cluster's center point, or *centroid*. As a result, different runs of *k*-means—each with a randomized start—will result in slightly different clustering assignments of the observations. From these different runs, we can choose the one that has the best separation, defined as the lowest total sum of within-cluster variations across all *k* clusters.[7]

Hierarchical clustering

An alternative clustering approach—one that does not require us to precommit to a particular number of clusters—is known as *hierarchical clustering*. One version of hierarchical clustering called *agglomerative clustering* uses a tree-based clustering method, and builds what is called a *dendrogram*. A dendrogram can be depicted graphically as an upside-down tree, where the leaves are at the bottom and the tree trunk is at the top.

7 There are faster variants of *k*-means clustering such as mini-batch *k*-means, which we cover later in the book.

The leaves at the very bottom are individual instances in the dataset. Hierarchical clustering then joins the leaves together—as we move vertically up the upside-down tree—based on how similar they are to each other. The instances (or groups of instances) that are most similar to each other are joined sooner, while the instances that are not as similar are joined later. With this iterative process, all the instances are eventually linked together forming the single trunk of the tree.

This vertical depiction is very helpful. Once the hierarchical clustering algorithm has finished running, we can view the dendrogram and determine where we want to cut the tree—the lower we cut, the more individual branches we are left with (i.e., more clusters). If we want fewer clusters, we can cut higher on the dendrogram, closer to the single trunk at the very top of this upside-down tree. The placement of this vertical cut is similar to choosing the number of k clusters in the k-means clustering algorithm.[8]

DBSCAN

An even more powerful clustering algorithm (based on the density of points) is known as *DBSCAN* (density-based spatial clustering of applications with noise). Given all the instances we have in space, DBSCAN will group together those that are packed closely together, where close together is defined as a minimum number of instances that must exist within a certain distance. We specify both the minimum number of instances required and the distance.

If an instance is within this specified distance of multiple clusters, it will be grouped with the cluster to which it is most densely located. Any instance that is not within this specified distance of another cluster is labeled an outlier.

Unlike k-means, we do not need to prespecify the number of clusters. We can also have arbitrarily shaped clusters. DBSCAN is much less prone to the distortion typically caused by outliers in the data.

Feature Extraction

With unsupervised learning, we can learn new representations of the original features of data—a field known as *feature extraction*. Feature extraction can be used to reduce the number of original features to a smaller subset, effectively performing dimensionality reduction. But feature extraction can also generate new feature representations to help improve performance on supervised learning problems.

8 Hierarchical clustering uses Euclidean distance by default, but it can also use other similarity metrics such as correlation-based distance, which we will explore in greater detail later in the book.

Autoencoders

To generate new feature representations, we can use a feedforward, nonrecurrent neural network to perform representation learning, where the number of nodes in the output layer matches the number of nodes in the input layer. This neural network is known as an *autoencoder* and effectively reconstructs the original features, learning a new representation using the hidden layers in between.[9]

Each hidden layer of the autoencoder learns a representation of the original features, and subsequent layers build on the representation learned by the preceding layers. Layer by layer, the autoencoder learns increasingly complicated representations from simpler ones.

The output layer is the final newly learned representation of the original features. This learned representation can then be used as an input into a supervised learning model with the objective of improving the generalization error.

Feature extraction using supervised training of feedforward networks

If we have labels, an alternate feature extraction approach is to use a feedforward, nonrecurrent neural network where the output layer attempts to predict the correct label. Just like with autoencoders, each hidden layer learns a representation of the original features.

However, when generating the new representations, this network is explicitly *guided by the labels*. To extract the final newly learned representation of the original features in this network, we extract the penultimate layer—the hidden layer just before the output layer. This penultimate layer can then be used as an input into any supervised learning model.

Unsupervised Deep Learning

Unsupervised learning performs many important functions in the field of deep learning, some of which we will explore in this book. This field is known as *unsupervised deep learning*.

Until very recently, the training of deep neural networks was computationally intractable. In these neural networks, the hidden layers learn internal representations to help solve the problem at hand. The representations improve over time based on how the neural network uses the *gradient of the error function* in each training iteration to update the weights of the various nodes.

9 There are several types of autoencoders, and each learns a different set of representations. These include denoising autoencoders, sparse autoencoders, and variational autoencoders, all of which we will explore later in the book.

These updates are computationally expensive, and two major types of problems may occur in the process. First, the gradient of the error function may become very small, and, since *backpropagation* relies on multiplying these small weights together, the weights of the network may update very slowly or not at all, preventing proper training of the network.[10] This is known as the *vanishing gradient problem*.

Conversely, the other issue is that the gradient of the error function might become very large; with backprop, the weights throughout the network may update in huge increments, making the training of the network very unstable. This is known as the *exploding gradient problem*.

Unsupervised pretraining

To address these difficulties in training very deep, multilayered neural networks, machine learning researchers train neural networks in multiple, successive stages, where each stage involves a shallow neural network. The output of one shallow network is then used as the input of the next neural network. Typically, the first shallow neural network in this pipeline involves an unsupervised neural network, but the later networks are supervised.

This unsupervised portion is known as *greedy layer-wise unsupervised pretraining*. In 2006, Geoffrey Hinton demonstrated the successful application of unsupervised pretraining to initialize the training of deeper neural network pipelines, kicking off the current deep learning revolution. Unsupervised pretaining allows the AI to capture an improved representation of the original input data, which the supervised portion then takes advantage of to solve the specific task at hand.

This approach is called "greedy" because each portion of the neural network is trained independently, not jointly. "Layer-wise" refers to the layers of the network. In most modern neural networks, pretraining is usually not necessary. Instead, all the layers are trained jointly using backpropagation. Major computer advances have made the vanishing gradient problem and the exploding gradient problem much more manageable.

Unsupervised pretraining not only makes supervised problems easier to solve but also facilitates *transfer learning*. Transfer learning involves using machine learning algorithms to store knowledge gained from solving one task to solve another related task much more quickly and with considerably less data.

10 Backpropagation (also known as *backward propagation of errors*) is a gradient descent-based algorithm used by neural networks to update weights. In backprop, the weights of the final layer are calculated first and then used to update the weights of the preceding layers. This process continues until the weights of the very first layer are updated.

Restricted Boltzmann machines

One applied example of unsupervised pretraining is the *restricted Boltzmann machine (RBM)*, a shallow, two-layer neural network. The first layer is the input layer, and the second layer is the hidden layer. Each node is connected to every node in the other layer, but nodes are not connected to nodes of the same layer—this is where the restriction occurs.

RBMs can perform unsupervised tasks such as dimensionality reduction and feature extraction and provide helpful unsupervised pretraining as part of supervised learning solutions. RBMs are similar to autoencoders but differ in some important ways. For example, autoencoders have an output layer, while RBMs do not. We will explore these and other differences in detail later in the book.

Deep belief networks

RBMs can be linked together to form a multistage neural network pipeline known as a *deep belief network (DBN)*. The hidden layer of each RBM is used as the input for the next RBM. In other words, each RBM generates a representation of the data that the next RBM then builds upon. By successively linking this type of representation learning, the deep belief network is able to learn more complicated representations that are often used as *feature detectors*.[11]

Generative adversarial networks

One major advance in unsupervised deep learning has been the advent of *generative adversarial networks (GANs)*, introduced by Ian Goodfellow and his fellow researchers at the University of Montreal in 2014. GANs have many applications; for example, we can use GANs to create near-realistic synthetic data, such as images and speech, or perform anomaly detection.

In GANs, we have two neural networks. One network—known as the generator—generates data based on a model data distribution it has created using samples of real data it has received. The other network—known as the discriminator—discriminates between the data created by the generator and data from the true data distribution.

As a simple analogy, the generator is the counterfeiter, and the discriminator is the police trying to identify the forgery. The two networks are locked in a zero-sum game. The generator is trying to fool the discriminator into thinking the synthetic data comes from the true data distribution, and the discriminator is trying to call out the synthetic data as fake.

11 Feature detectors learn good representations of the original data, helping separate distinct elements. For example, in images, feature detectors help separate elements such as noses, eyes, mouths, etc.

GANs are unsupervised learning algorithms because the generator can learn the underlying structure of the true data distribution even when there are no labels. GANs learn the underlying structure in the data through the training process and efficiently capture the structure using a small, manageable number of parameters.

This process is similar to the representation learning that occurs in deep learning. Each hidden layer in the neutral network of a generator captures a representation of the underlying data—starting very simply—and subsequent layers pick up more complicated representations by building on the simpler preceding layers.

Using all these layers together, the generator learns the underlying structure of the data and, using what it has learned, the generator attempts to create synthetic data that is nearly identical to the true data distribution. If the generator has captured the essence of the true data distribution, the synthetic data will appear real.

Sequential Data Problems Using Unsupervised Learning

Unsupervised learning can also handle sequential data such as time series data. One such approach involves learning the hidden states of a *Markov model*. In the *simple Markov model*, states are fully observed and change stochastically (in other words, randomly). Future states depend only on the current state and are not dependent on previous states.

In a *hidden Markov model*, the states are only partially observable, but, like with simple Markov models, the outputs of these partially observable states are fully observable. Since the observations that we have are insufficient to determine the state completely, we need unsupervised learning to help discover these hidden states more fully.

Hidden Markov model algorithms involve learning the probable next state given what we know about the sequence of previously occurring, partially observable states and fully observable outputs. These algorithms have had major commercial applications in sequential data problems involving speech, text, and time series.

Reinforcement Learning Using Unsupervised Learning

Reinforcement learning is the third major branch of machine learning, in which an *agent* determines its optimal behavior (*actions*) in an *environment* based on feedback (*reward*) that it receives. This feedback is known as the *reinforcement signal*. The agent's goal is to maximize its cumulative reward over time.

While reinforcement learning has been around since the 1950s, it has made mainstream headline news only in recent years. In 2013, DeepMind—now owned by Google—applied reinforcement learning to achieve superhuman-level performance at

playing many different Atari games. DeepMind's system achieved this with just raw sensory data as input and no prior knowledge of the rules of the games.

In 2016, DeepMind again captured the imagination of the machine learning community—this time the DeepMind reinforcement learning-based AI agent AlphaGo beat Lee Sedol, one of the world's best Go players. These successes have cemented reinforcement learning as a mainstream AI topic.

Today, machine learning researchers are applying reinforcement learning to solve many different types of problems including:

- Stock market trading, in which the agent buys and sells (actions) and receives profits or losses (rewards) in return
- Video games and board games, in which the agent makes game decisions (actions) and wins or loses (rewards)
- Self-driving cars, in which the agent directs the vehicle (actions) and either stays on course or crashes (rewards)
- Machine control, in which the agent moves about its environment (actions) and either completes the course or fails (rewards)

In the simplest reinforcement learning problems, we have a finite problem—with a finite number of states of the environment, a finite number of actions that are possible at any given state of the environment, and a finite number of rewards. The action taken by the agent given the current state of the environment determines the next state, and the agent's goal is to maximize its long-term reward. This family of problems is known as finite *Markov decision processes*.

However, in the real world, things are not so simple—the reward is unknown and dynamic rather than known and static. To help discover this unknown reward function and approximate it as best as possible, we can apply unsupervised learning. Using this approximated reward function, we can apply reinforcement learning solutions to increase the cumulative reward over time.

Semisupervised Learning

Even though supervised learning and unsupervised learning are two distinct major branches of machine learning, the algorithms from each branch can be mixed together as part of a machine learning pipeline.[12] Typically, this mix of supervised and unsupervised is used when we want to take full advantage of the few labels that we have or when we want to find new, yet unknown patterns from unlabeled data in

12 Pipeline refers to a system of machine learning solutions that are applied in succession to achieve a larger objective.

addition to the known patterns from the labeled data. These types of problems are solved using a hybrid of supervised and unsupervised learning known as semisupervised learning. We will explore this area in greater detail later in the book.

Successful Applications of Unsupervised Learning

In the last ten years, most successful commercial applications of machine learning have come from the supervised learning space, but this is changing. Unsupervised learning applications have become more commonplace. Sometimes, unsupervised learning is just a means to make supervised applications better. Other times, unsupervised learning achieves the commercial application itself. Here is a closer look at two of the biggest applications of unsupervised learning to date: anomaly detection and group segmentation.

Anomaly Detection

Performing dimensionality reduction can reduce the original high-dimensional feature space into a transformed lower-dimensional space. In this lower-dimensional space, we find where the majority of points densely lie. This portion is the *normal space*. Points that lie much farther away are called *outliers*—or *anomalies*—and are worth investigating in greater detail.

Anomaly detection systems are commonly used for fraud detection such as credit card fraud, wire fraud, cyber fraud, and insurance fraud. Anomaly detection is also used to identify rare, malicious events such as hacking of internet-connected devices, maintenance failures in mission-critical equipment such as airplanes and trains, and cybersecurity breaches due to malware and other pernicious agents.

We can use these systems for spam detection, such as the email spam filter example we used earlier in the chapter. Other applications include finding bad actors to stop activity such as terrorist financing, money laundering, human and narcotics trafficking, and arms dealing, identifying high risk events in financial trading, and discovering diseases such as cancer.

To make the analysis of anomalies more manageable, we can use a clustering algorithm to group similar anomalies together and then hand-label these clusters based on the types of behavior they represent. With such a system, we can have an unsupervised learning AI that is able to identify anomalies, cluster them into appropriate groups, and, using the cluster labels provided by humans, recommend to business analysts the appropriate course of action.

With anomaly detection systems, we can take an unsupervised problem and eventually create a semisupervised one with this cluster-and-label approach. Over time, we can run supervised algorithms on the labeled data alongside the unsupervised

algorithms. For successful machine learning applications, unsupervised systems and supervised systems should be used in conjunction, complementing one another.

The supervised system finds the known patterns with a high level of accuracy, while the unsupervised system discovers new patterns that may be of interest. Once these patterns are uncovered by the unsupervised AI, the patterns are labeled by humans, transitioning more of the data from unlabeled to labeled.

Group segmentation

With clustering, we can segment groups based on similarity in behavior in areas such as marketing, customer retention, disease diagnosis, online shopping, music listening, video watching, online dating, social media activity, and document classification. The amount of data that is generated in each of these areas is massive, and the data is only partially labeled.

For patterns that we already know and want to reinforce, we can use supervised learning algorithms. But often we want to discover new patterns and groups of interest—for this discovery process, unsupervised learning is a natural fit. Again, it is all about synergy. We should use supervised and unsupervised learning systems in conjunction to build a stronger machine learning solution.

Conclusion

In this chapter, we explored the following:

- The difference between a rules-based system and machine learning
- The difference between supervised and unsupervised learning
- How unsupervised learning can help address common problems in training machine learning models
- Common algorithms for supervised, unsupervised, reinforcement, and semisupervised learning
- Two major applications of unsupervised learning—anomaly detection and group segmentation

In Chapter 2, we'll explore how to build machine learning applications. Then, we will cover dimensionality reduction and clustering in detail, building an anomaly detection system and a group segmentation system in the process.

O'REILLY®

Hands-On Unsupervised Learning Using Python

Many industry experts consider unsupervised learning the next frontier in artificial intelligence, one that may hold the key to general artificial intelligence. Since the majority of the world's data is unlabeled, conventional supervised learning cannot be applied. Unsupervised learning, on the other hand, can be applied to unlabeled datasets to discover meaningful patterns buried deep in the data, patterns that may be near impossible for humans to uncover.

Author Ankur Patel shows you how to apply unsupervised learning using two simple, production-ready Python frameworks: Scikit-learn and TensorFlow using Keras. With code and hands-on examples, data scientists will identify difficult-to-find patterns in data and gain deeper business insight, detect anomalies, perform automatic feature engineering and selection, and generate synthetic datasets. All you need is programming and some machine learning experience to get started.

- **Compare the strengths and weaknesses of the different machine learning approaches: supervised, unsupervised, and reinforcement learning**
- **Set up and manage machine learning projects end-to-end**
- **Build an anomaly detection system to catch credit card fraud**
- **Clusters users into distinct and homogeneous groups**
- **Perform semi-supervised learning**
- **Develop movie recommender systems using restricted Boltzmann machines**
- **Generate synthetic images using generative adversarial networks**

"Researchers, engineers, and students will appreciate this book full of practical, unsupervised learning techniques written in plain English, with straightforward Python examples that can be implemented quickly and effectively."

-Sarah Nagy
Senior Data Scientist at Edison

Ankur A. Patel is the vice president of data science at 7Park Data, a Vista Equity Partners portfolio company. At 7Park Data, Ankur and his data science team use alternative data to build data products for hedge funds and corporations and develop machine learning as a service (MLaaS) for enterprise clients.

PYTHON

MRP: ₹ 1,225 .00

Twitter: @oreillymedia
facebook.com/oreilly

SPD®
SHROFF PUBLISHERS & DISTRIBUTORS PVT. LTD.

ISBN : 978-93-5213-812-8

9 789352 138128

First Edition/2019/Paperback/English