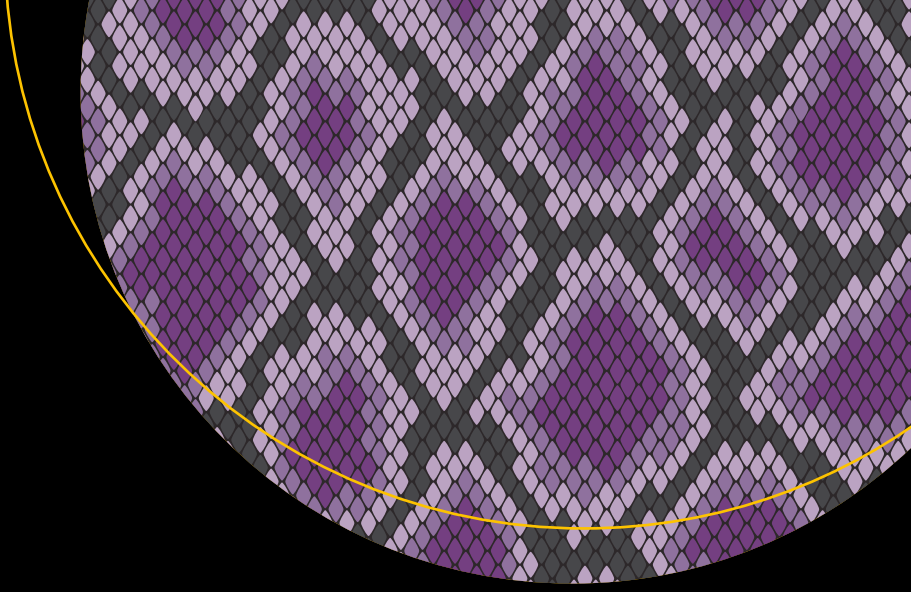

Deep Learning with Python

Learn Best Practices of Deep Learning Models with PyTorch

—

Second Edition

—

Nikhil Ketkar
Jojo Moolayil

Apress®

Deep Learning with Python

Learn Best Practices of Deep Learning Models with PyTorch

Second Edition

Nikhil Ketkar
Jojo Moolayil

Apress®

Deep Learning with Python: Learn Best Practices of Deep Learning Models with PyTorch

Nikhil Ketkar
Bangalore, Karnataka, India

Jojo Moolayil
Vancouver, BC, Canada

ISBN-13 (pbk): 978-1-4842-5363-2
ISBN-13 (electronic): 978-1-4842-5364-9
https://doi.org/10.1007/978-1-4842-5364-9

Copyright © 2021 by Nikhil Ketkar, Jojo Moolayil

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Celestin Suresh John
Development Editor: James Markham
Coordinating Editor: Aditee Mirashi

Cover designed by eStudioCalamar

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-5363-2. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

Table of Contents

About the Authors

Nikhil Ketkar currently leads the Machine Learning Platform team at Flipkart, India's largest ecommerce company. He received his PhD from Washington State University. Following that, he conducted postdoctoral research at University of North Carolina at Charlotte, which was followed by a brief stint in high-frequency trading at TransMarket in Chicago. More recently, he led the data mining team at Guavus, a startup doing big data analytics in the telecom domain, and Indix, a startup doing data science in the ecommerce domain. His research interests include machine learning and graph theory.

Jojo Moolayil is an artificial intelligence professional and published author of three books on machine learning, deep learning, and IoT. He is currently working with Amazon Web Services as a Research Scientist – A.I. in their Vancouver, BC office.

In his current role with AWS, Jojo works on researching and developing large-scale A.I. solutions for combating fraud and enriching the customer's payment experience in the cloud. He is also actively involved as a technical reviewer and AI consultant with leading publishers and has reviewed over a dozen books on machine learning, deep learning, and business analytics.

You can reach Jojo at:

- https://www.jojomoolayil.com/
- https://www.linkedin.com/in/jojo62000
- https://twitter.com/jojo62000

About the Technical Reviewers

Judy T. Raj is a Google Certified Professional Cloud Architect. She has great experience with the three leading cloud platforms—Amazon Web Services, Azure, and Google Cloud Platform—and has co-authored a book on Google Cloud Platform with Packt Publications. She has also worked with a wide range of technologies in machine learning, data science, blockchains, IoT, robotics, and mobile and web app development. She is currently a technical content engineer in Loonycorn. Judy holds a degree in computer science and engineering from Cochin University of Science and Technology. A driven engineer fascinated with technology, she is a passionate coder, a machine language enthusiast, and a blockchain aficionado.

Manohar Swamynathan is a data science practitioner and an avid programmer, with more than 14 years of experience in various data science-related areas, including data warehousing, business intelligence (BI), analytical tool development, ad-hoc analysis, predictive modeling, data science product development, consulting, formulating strategy, and executing analytics programs.

His career has covered the life cycle of data across multiple domains, such as US mortgage banking, retail/ecommerce, insurance, and industrial IoT. Manohar has a bachelor's degree with a specialization in physics, mathematics, computers, and a master's degree in project management. He is currently living in Bengaluru, the silicon valley of India.

Acknowledgments

I would like to thank my colleagues at Flipkart and Indix, and the technical reviewers, for their feedback and comments. I will also like to thank Charu Mudholkar for proofreading the book in its final stages.

—Nikhil Ketkar

I would like to thank my beloved wife, Divya, for her constant support.

—Jojo Moolayil

Introduction

This book has been drafted with a unique approach. The second edition focuses on the practicality of the topics within deep learning that help the reader to embrace modern tools with the right mathematical foundations. The first edition focused on introducing a meaningful foundation for the subject, while limiting the depth of the practical implementations. While we explored a breadth of technical frameworks for deep learning (Theano, TensorFlow, Keras, and PyTorch), we limited the depth of the implementation details. The idea was to distill the mathematical foundations while focusing briefly on the practical tools used for implementation.

A lot has changed over the past three years. The deep learning fraternity is now stronger than ever, and the frameworks have evolved in size and adoption. Theano is now deprecated (ceased development); TensorFlow saw huge adoption in the industry and academia; and Keras became more popular among beginners and deep learning enthusiasts. However, PyTorch has emerged recently as a widely popular choice for academia as well as industry. The growing number of research publications that recently have used PyTorch over TensorFlow is a testament to its growth within deep learning.

On the same note, we felt the need to revise the book with a focus on engaging readers with hands-on exercises to aid a more meaningful understanding of the subject. In this book, we have struck the perfect balance, with mathematical foundations as well as hands-on exercises, to embrace practical implementation exclusively on PyTorch. Each exercise is supplemented with the required explanations of PyTorch's functionalities and required abstractions for programming complexities.

Part I serves as a brief introduction to machine learning, deep learning, and PyTorch. We explore the evolution of the field, from early rule-based systems to the present-day sophisticated algorithms, in an accelerated fashion.

Part II explores the essential deep learning building blocks. Chapter 3 introduces a simple feed-forward neural network. Incrementally and logically, we uncover the various building blocks that constitute a neural network and which can be reused in building any other network. Though foundational, Chapter 3 focuses on building a baby neural network with the required framework that helps to construct and train networks of all kinds and complexities. In Chapter 4, we explore the core idea that enabled the possibility of training large networks through backpropagation using automatic differentiation and chain rule. We explore PyTorch's Autograd module with a small example to understand how the solution works programmatically. In Chapter 5, we look at orchestrating all the building blocks discussed through so far, along with the performance metrics of deep learning models and the artifacts required to enable an improved means for training—i.e., regularization, hyperparameter tuning, overfitting, underfitting, and model capacity. Finally, we leverage all this content to develop a deep neural network for a real-life dataset using PyTorch. In this exercise, we also explore additional PyTorch constructs that help in the orchestration of various deep learning building blocks.

Part III covers three important topics within deep learning. Chapter 6 explores convolutional neural networks and introduces the field of computer vision. We explore the core topics within convolutional neural networks, including how they learn and how they are distinguished from other networks. We also leverage a few hands-on exercises—using a small MNIST dataset as well as the popular Cats and Dogs dataset—to study the practical implementation of a convolutional neural network. In Chapter 7, we study recurrent neural networks and enter the field of natural language processing. Similar to Chapter 6, we incrementally build an intuition

around the fundamentals and later explore practical exercises with real-life datasets. Chapter 8 concludes the book by looking at some of the recent trends within deep learning. This chapter is only a cursory introduction and does not include any implementation details. The objective is to highlight some advances in the research and the possible next steps for advanced topics.

Overall, we have put in great efforts to write a structured, concise, exercise-rich book that balances the coverage between the mathematical foundations and the practical implementation.

CHAPTER 1

Introduction to Machine Learning and Deep Learning

The subject of deep learning has gained immense popularity recently, and, in the process, has given rise to several terminologies that make distinguishing them fairly complex. One might find the task of neatly separating each field overwhelming, with the sheer volume of overlap between the topics.

This chapter introduces the subject of deep learning by discussing its historical context and how the field evolved into its present-day form. Later, we will introduce machine learning by covering the foundational topics in brief. To start with deep learning, we will leverage the constructs gained from machine learning using basic Python. Chapter 2 begins the practical implementation using PyTorch.

Defining Deep Learning

Deep learning is a subfield within machine learning that deals with the algorithms that closely resemble an over-simplified version of the human brain that solves a vast category of modern-day machine intelligence. Many common examples can be found within the smartphone's app

© Nikhil Ketkar, Jojo Moolayil 2021
N. Ketkar and J. Moolayil, *Deep Learning with Python*,
https://doi.org/10.1007/978-1-4842-5364-9_1

ecosystem (iOS and Android): face detection on the camera, auto-correct and predictive text on keyboards, AI-enhanced beautification apps, smart assistants like Siri/Alexa/Google Assistant, Face-ID (face unlock on iPhones), video suggestions on YouTube, friend suggestions on Facebook, cat filters on Snapchat are all products that were made the state-of-the-art only for deep learning. Essentially, deep learning is ubiquitous in the today's digital life.

Truth be told, it can be complicated to define deep learning without navigating some historical context.

A Brief History

The journey of artificial intelligence (AI) to its present day can be broadly divided into four parts: viz. rule-based systems, knowledge-based systems, machine, and deep learning. Although the granular transitions in the journey can be mapped into several important milestones, we will cover a more simplistic overview. The entire evolution is encompassed into the larger idea of "artificial intelligence." Let's take a step-by-step approach to tackle this broad term.

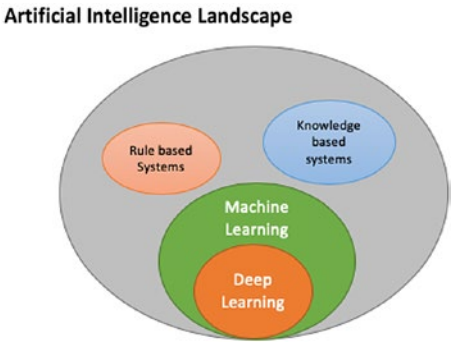

Figure 1-1. *The AI landscape*

The journey of Deep Learning starts with the field of artificial intelligence, the rightful parent of the field, and has a rich history going back to the 1950s. The field of artificial intelligence can be defined in simple terms as the ability of machines to think and learn. In more layman words, we would define it as the process of aiding machines with intelligence in some form so that they can execute a task better than before. The above Figure 1-1 showcases a simplified landscape of AI with the various aforementioned fields showcased a subset. We will explore each of these subsets in more detail in the section below.

Rule-Based Systems

The intelligence we induce into a machine may not necessarily be a sophisticated process or ability; something as simple as a set of rules can be defined as intelligence. The first-generation AI products were simply rule-based systems, wherein a comprehensive set of rules were guided to the machine to map the exhaustive possibilities. A machine that executes a task based on defined rules would result in a more appealing outcome than a rigid machine (one without intelligence).

A more layman example for the modern-day equivalent would be an ATM that dispenses cash. Once authenticated, users enter the amount they want and the machine, based on the existing combination of notes in-store, dispenses the correct amount with the least number of bills. The logic (intelligence) for the machine to solve the problem is explicitly coded (designed). The designer of the machine carefully thought through the comprehensive list of possibilities and designed a system that can solve the task programmatically with finite time and resources.

Most of the early day's success in artificial intelligence was fairly simple. Such tasks can be easily described formally, like the game of checkers or chess. This notion of *being able to easily describe the task formally* is at the heart of what can or cannot be done easily by a computer program. For instance, consider the game of chess. The

formal description of the game of chess would be the representation of the board, a description of how each of the pieces moves, the starting configuration, and a description of the configuration wherein the game terminates. With these notions formalized, it is relatively easy to model a chess-playing AI program as a search, and, given sufficient computational resources, it's possible to produce relatively good chess-playing AI.

The first era of AI focused on such tasks with a fair amount of success. At the heart of the methodology were a symbolic representation of the domain and the manipulation of the symbols based on given rules (with increasingly sophisticated algorithms for searching the solution space to arrive at a solution).

It must be noted that the formal definitions of such rules were done manually. However, such early AI systems were fairly general-purpose task/problem solvers in the sense that any problem that could be described formally could be solved with the generic approach.

The key limitation of such systems is that the game of chess is a relatively easy problem for AI simply because the problem set is relatively simple and can be easily formalized. This is not the case with many of the problems human beings solve on a day-to-day basis (natural intelligence). For instance, consider diagnosing a disease or transcribing human speech to text. These tasks, which human beings can do but which are hard to describe formally, presented as a challenge in the early days of AI.

Knowledge-Based Systems

The challenge of addressing natural intelligence to solve day-to-day problems evolved the landscape of AI into an approach akin to human-beings—i.e., by leveraging a large amount of knowledge about the task/problem domain. Given this observation, subsequent AI systems relied on large knowledge bases that captured the knowledge about the problem/task domain. Note that the term used here is *knowledge*, not *information* or *data*. By knowledge, we simply mean data/information that a program/algorithm can reason about. An example could be a graph representation

of a map with edges labeled with distances and about of traffic (which is being constantly updated), allowing a program to reason about the shortest path between points.

Such knowledge-based systems, wherein the knowledge was compiled by experts and represented in a way that allowed algorithms/programs to reason about it, represented the second generation of AI. At the heart of such approaches were increasingly sophisticated approaches for representing and reasoning about knowledge to solve tasks/problems that required such knowledge. Examples of such sophistication include the use of first-order logic to encode knowledge and probabilistic representations to capture and reason where uncertainty is inherent to the domain.

One of the key challenges that such systems faced, and addressed to some extent, was the uncertainty inherent in many domains. Human beings are relatively good at reasoning in environments with unknowns and uncertainty. One key observation here is that even the knowledge we hold about a domain is not black or white but grey. A lot of progress was made in this era on representing and reasoning about unknowns and uncertainty. There were some limited successes in tasks like diagnosing a disease that relied on leveraging and reasoning using a knowledge base in the presence of unknowns and uncertainty.

The key limitation of such systems was the need to hand-compile the knowledge about the domain from experts. Collecting, compiling, and maintaining such knowledge bases rendered such systems impractical. In certain domains, it was extremely hard to even collect and compile such knowledge—for example, transcribing speech to text or translating documents from one language to another. While human beings can easily learn to do such tasks, it's extremely challenging to hand-compile and encode the knowledge related to the tasks—for instance, the knowledge of the English language and grammar, accents, and subject matter. To address these challenges, machine learning is the way forward.

Machine Learning

In formal terms, we define machine learning as the field within AI where intelligence is added without explicit programming. Human beings acquire knowledge for any task through learning. Given this observation, the focus of subsequent work in AI shifted over a decade or two to algorithms that improved their performance based on data provided to them. The focus of this subfield was to develop algorithms that acquired relevant knowledge for a task/problem domain given data. It is important to note that this knowledge acquisition relied on labeled data and a suitable representation of labeled data as defined by a human being.

Consider, for example, the problem of diagnosing a disease. For such a task, a human expert would collect a lot of cases where a patient had and did not have the disease in question. Then, the human expert would identify a number of features that would aid in making the prediction—for example, the age and gender of the patient, and the results from a number of diagnostic tests, such as blood pressure, blood sugar, etc. The human expert would compile all this data and represent it in a suitable form—for example, by scaling/normalizing the data, etc. Once this data were prepared, a machine learning algorithm could learn how to infer whether the patient has the disease or not by generalizing from the labeled data. Note that the labeled data consisted of patients that both have and do not have the disease. So, in essence, the underlying machine language algorithm is essentially doing the job of finding a mathematical function that can produce the right outcome (disease or no disease) given the inputs (features like age, gender, data from diagnostic tests, and so forth). Finding the simplest mathematical function that predicts the outputs with the required level of accuracy is at the heart of the field of machine learning. For example, questions related to the number of examples required to learn a task or the time complexity of an algorithm are specific areas for which the field of ML has provided answers with theoretical justification. The field has matured to a point where, given enough data,

compute resources, and human resources to engineer features, a large class of problems are solvable.

The key limitation of mainstream machine language algorithms is that applying them to a new problem domain requires a massive amount of feature engineering. For instance, consider the problem of recognizing objects in images. Using traditional machine language techniques, such a problem would require a massive feature-engineering effort wherein experts identify and generate features that would be used by the machine language algorithm. In a sense, true intelligence is in the identification of features; the machine language algorithm is simply learning how to combine these features to arrive at the correct answer. This identification of features or the representation of data that domain experts do before machine language algorithms are applied is both a conceptual and practical bottleneck in AI.

It's a conceptual bottleneck because if features are being identified by domain experts and the machine language algorithm is simply learning to combine and draw conclusions from this, is this really AI? It's a practical bottleneck because the process of building models via traditional machine language is bottlenecked by the amount of feature engineering required. There are limits to how much human effort can be thrown at the problem.

Deep Learning

The major bottleneck in machine learning systems was solved with deep learning. Here, we essentially took the intelligence one step further, where the machine develops relevant features for the task in an automated way instead of hand-crafting. Human beings learn concepts starting from raw data. For instance, a child shown with a few examples of a particular animal (say, cats) will soon learn to identify the animal. The learning process does not involve a parent identifying a cat's features, such as its whiskers, fur, or tail. Human learning goes from raw data to a conclusion without the explicit step where features are identified and provided to the learner. In a sense, human beings learn the appropriate representation

of data from the data itself. Furthermore, they organize concepts as a hierarchy where complicated concepts are expressed using primitive concepts.

The field of deep learning has its primary focus on learning appropriate representations of data such that these could be used to conclude. The word "deep" in "deep learning" refers to the idea of learning the hierarchy of concepts directly from raw data. A more technically appropriate term for deep learning would be *representation learning*, and a more practical term for the same would be *automated feature engineering*.

Advances in Related Fields

It is important to note the advances in other fields like compute power, storage cost, etc. that have played a key role in the recent interest and success of deep learning. Consider the following, for example:

- The ability to collect, store and process large amounts of data has greatly advanced over the last decade (for instance, the Apache Hadoop ecosystem).
- The ability to generate supervised training data (data with labels—for example, pictures annotated with the objects in the picture) has improved a lot with the availability of crowd-sourcing services (like Amazon Mechanical Turk).
- The massive improvements in computational horsepower brought about by graphical processing units (GPUs) enabled parallel computing to new heights.
- The advances in both the theory and software implementation of automatic differentiation (such as PyTorch or Theano) accelerated the speed of development and research for deep learning.

Although these advancements are peripheral to deep learning, they have played a big role in enabling advances in deep learning.

Prerequisites

The key prerequisites for reading this book include a working knowledge of Python and some coursework in linear algebra, calculus, and probability. Readers should refer to the following in case they need to cover these prerequisites.

- *Dive Into Python*, by Mark Pilgrim - Apress Publications (2004)
- *Introduction to Linear Algebra (Fifth Edition)*, by Gilbert Strang - Wellesley-Cambridge Press
- *Calculus*, by Gilbert Strang - Wellesley-Cambridge Press
- *All of Statistics* (Section 1, chapters 1-5), by Larry Wasserman - Springer (2010)

The Approach Ahead

This book focuses on the key concepts of deep learning and its practical implementation using PyTorch. In order to use PyTorch, you should possess a basic understanding of Python programming. Chapter 2 introduces PyTorch, and the subsequent chapters discuss additional important constructs within PyTorch.

Before delving into deep learning, we need to discuss the basic constructs of machine learning. In the remainder of this chapter, we will explore the baby steps of machine learning with a dummy example. To implement the constructs, we will use Python and again implement the same using PyTorch.

Installing the Required Libraries

You need to install a number of libraries in order to run the source code for the examples in this book. We recommend installing the Anaconda Python distribution (https://www.anaconda.com/products/individual), which simplifies the process of installing the required packages (using either conda or pip). The list of packages you need include NumPy, matplotlib, scikit-learn, and PyTorch.

PyTorch is not installed as a part of the Anaconda distribution. You should install PyTorch, torchtext, and torchvision, along with the Anaconda environment.

Note that Python 3.6 (and above) is recommended for the exercises in this book. We highly recommend creating a new Python environment after installing the Anaconda distribution.

Create a new environment with Python 3.6 (use Terminal in Linux/Mac or the Command Prompt in Windows), and then install the additional necessary packages, as follows:

```
conda create -n testenvironment python=3.6

conda activate testenvironment
pip install pytorch torchvision torchtext
```

For additional help with PyTorch, please refer to the Get Started guide at https://pytorch.org/get-started/locally/.

The Concept of Machine Learning

As human beings, we are intuitively aware of the concept of learning. It simply means to get better at a task over time. The task could be physical, such as learning to drive a car, or intellectual, such as learning a new language. The subject of machine learning focuses on the development of algorithms that can learn as humans learn; that is, they get better at a task

over a period over time and with experience—thus inducing intelligence without explicit programming.

The first question to ask is why we would be interested in the development of algorithms that improve their performance over time, with experience. After all, many algorithms are developed and implemented to solve real-world problems that don't improve over time; they simply are developed by humans, implemented in software, and get the job done. From banking to ecommerce and from navigation systems in our cars to landing a spacecraft on the moon, algorithms are everywhere, and, a majority of them do not improve over time. These algorithms simply perform the task they are intended to perform, with some maintenance required from time to time. Why do we need machine learning?

The answer to this question is that for certain tasks it is easier to develop an algorithm that learns/improves its performance with experience than to develop an algorithm manually. Although this might seem unintuitive to the reader at this point, we will build intuition for this during this chapter.

Machine learning can be broadly classified as *supervised learning*, where training data with labels is provided for the model to learn, and *unsupervised learning*, where the training data lacks labels. We also have *semi-supervised learning* and *reinforcement learning*, but for now, we would limit our scope to supervised machine learning. Supervised learning can again be classified into two areas: *classification*, for discrete outcomes, and *regression*, for continuous outcomes.

Binary Classification

In order to further discuss the matter at hand, we need to be precise about some of the terms we have been intuitively using, such as task, learning, experience, and improvement. We will start with the task of binary classification.

Consider an abstract problem domain where we have data of the form

$$D = \{(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)\}$$

where $x \in \mathbb{R}^n$ and $y = \pm 1$.

We do not have access to all such data but only a subset $S \in D$. Using S, our task is to generate a computational procedure that implements the function $f: x \rightarrow y$ such that we can use f to make predictions over unseen data $(x_i, y_i) \notin S$ that are correct, $f(x_i) = y_i$. Let's denote $U \in D$ as the set of unseen data—that is, $(x_i, y_i) \notin S$ and $(x_i, y_i) \in U$.

We measure performance over this task as the error over unseen data

$$E(f, D, U) = \frac{\sum_{(x_i, y_i) \in U} \left[f(x_i) \neq y_i \right]}{|U|}.$$

We now have a precise definition of the task, which is to categorize data into one of two categories ($y = \pm 1$) based on some seen data S by generating f. We measure performance (and improvement in performance) using the error $E(f, D, U)$ over unseen data U. The size of the seen data $|S|$ is the conceptual equivalent of experience. In this context, we want to develop algorithms that generate such functions f (which are commonly referred to as a model). In general, the field of machine learning studies the development of such algorithms that produce models that make predictions over unseen data for such, and, other formal tasks. (We introduce multiple such tasks later in the chapter.) Note that the x is commonly referred to as the *input/input variable* and y is referred to as the *output/output variable*.

As with any other discipline in computer science, the computational characteristics of such algorithms are an important facet; however, in addition to that, we also would like to have a model f that achieves a lower error $E(f, D, U)$ with as small a $|S|$ as possible.

Let's now relate this abstract but precise definition to a real-world problem so that our abstractions are grounded. Suppose that an ecommerce website wants to customize its landing page for registered

users to show the products they might be interested in buying. The website has historical data on users and would like to implement this as a feature to increase sales. Let's now see how this real-world problem maps on to the abstract problem of binary classification we described earlier.

The first thing that one might notice is that given a particular user and a particular product, one would want to predict whether the user will buy the product. Since this is the value to be predicted, it maps on to $y = \pm 1$, where we will let the value of $y = +1$ denote the prediction that the user will buy the product and the value of $y = -1$ denote the prediction that the user will not buy the product. Note that there is no particular reason for picking these values; we could have swapped this (let $y = +1$ denote the does not buy case and $y = -1$ denote the buy case), and there would be no difference. We just use $y = \pm 1$ to denote the two classes of interest to categorize data. Next, let's assume that we can represent the attributes of the product and the users buying and browsing history as $x \in \mathbb{R}^n$. This step is referred to as *feature engineering* in machine learning and we will cover it later in the chapter. For now, it suffices to say that we are able to generate such a mapping. Thus, we have historical data of what the users browsed and bought, attributes of a product, and whether the user bought the product or not mapped on to $\{(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)\}$. Now, based on this data, we would like to generate a function or a model $f: x \rightarrow y$, which we can use to determine which products a particular user will buy, and use this to populate the landing page for users. We can measure how well the model is doing on unseen data by populating the landing page for users, seeing whether they buy the products or not, and evaluating the error $E(f, D, U)$.

Regression

This section introduces another task: regression. Here, we have data of the form $D = \{(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)\}$, where $x \in \mathbb{R}^n$ and $y \in \mathbb{R}$, and our task is to generate a computational procedure that implements the function

$f: x \rightarrow y$. Note that instead of the prediction being a binary class label $y = \pm 1$, like in binary classification, we have real valued prediction. We measure performance over this task as the root-mean-square error (RMSE) over unseen data

$$E(f, D, U) = \left(\frac{\sum_{(x_i, y_i) \in U} \left(y_i - f(x_i) \right)^2}{|U|} \right)^{\frac{1}{2}}$$

Note that the RMSE is simply taking the difference between the predicted and actual value, squaring it so as to account for both positive and negative differences, taking the mean so as to aggregate over all the unseen data, and, finally, taking the square root so as to counterbalance the square operation.

A real-world problem that corresponds to the abstract task of regression is to predict the credit score for an individual based on their financial history, which can be used by a credit card company to extend the line of credit.

Generalization

Let's now cover what is the single most important intuition in machine leaning, which is that we want to develop/generate models that have good performance over unseen data. In order to do that, first will we introduce a toy data set for a regression task. Later, we will develop three different models using the same dataset with varying levels of complexity and study how the results differ to understand intuitively the concept of generalization.