Ranking, Recommender Systems, and Other Kinds of Learning

Because this is an introductory book, we focused on the most common machine learning tasks: classification and regression in supervised learning, and clustering and signal decomposition in unsupervised learning. There are many more kinds of machine learning out there, with many important applications. There are two particularly important topics that we did not cover in this book. The first is *ranking*, in which we want to retrieve answers to a particular query, ordered by their relevance. You've probably already used a ranking system today; this is how search engines operate. You input a search query and obtain a sorted list of answers, ranked by how relevant they are. A great introduction to ranking is provided in Manning, Raghavan, and Schütze's book *Introduction to Information Retrieval*. The second topic is *recommender systems*, which provide suggestions to users based on their preferences. You've probably encountered recommender systems under headings like "People You May Know," "Customers Who Bought This Item Also Bought," or "Top Picks for You." There is plenty of literature on the topic, and if you want to dive right in you might be interested in the now classic "Netflix prize challenge", in which the Netflix video streaming site released a large dataset of movie preferences and offered a prize of $1 million to the team that could provide the best recommendations. Another common application is prediction of time series (like stock prices), which also has a whole body of literature devoted to it. There are many more machine learning tasks out there—much more than we can list here—and we encourage you to seek out information from books, research papers, and online communities to find the paradigms that best apply to your situation.

Probabilistic Modeling, Inference, and Probabilistic Programming

Most machine learning packages provide predefined machine learning models that apply one particular algorithm. However, many real-world problems have a particular structure that, when properly incorporated into the model, can yield much better-performing predictions. Often, the structure of a particular problem can be expressed using the language of probability theory. Such structure commonly arises from having a mathematical model of the situation for which you want to predict. To understand what we mean by a structured problem, consider the following example.

Let's say you want to build a mobile application that provides a very detailed position estimate in an outdoor space, to help users navigate a historical site. A mobile phone provides many sensors to help you get precise location measurements, like the GPS, accelerometer, and compass. You also have an exact map of the area. This problem is highly structured. You know where the paths and points of interest are from your map. You also have rough positions from the GPS, and the accelerometer and compass in the user's device provide you with very precise relative measurements. But throwing these all together into a black-box machine learning system to predict positions might not be the best idea. This would throw away all the information you

already know about how the real world works. If the compass and accelerometer tell you a user is going north, and the GPS is telling you the user is going south, you probably can't trust the GPS. If your position estimate tells you the user just walked through a wall, you should also be highly skeptical. It's possible to express this situation using a probabilistic model, and then use machine learning or probabilistic inference to find out how much you should trust each measurement, and to reason about what the best guess for the location of a user is.

Once you've expressed the situation and your model of how the different factors work together in the right way, there are methods to compute the predictions using these custom models directly. The most general of these methods are called probabilistic programming languages, and they provide a very elegant and compact way to express a learning problem. Examples of popular probabilistic programming languages are `PyMC` (which can be used in Python) and `Stan` (a framework that can be used from several languages, including Python). While these packages require some understanding of probability theory, they simplify the creation of new models significantly.

Neural Networks

While we touched on the subject of neural networks briefly in Chapters 2 and 7, this is a rapidly evolving area of machine learning, with innovations and new applications being announced on a weekly basis. Recent breakthroughs in machine learning and artificial intelligence, such as the victory of the Alpha Go program against human champions in the game of Go, the constantly improving performance of speech understanding, and the availability of near-instantaneous speech translation, have all been driven by these advances. While the progress in this field is so fast-paced that any current reference to the state of the art will soon be outdated, the recent book *Deep Learning* by Ian Goodfellow, Yoshua Bengio, and Aaron Courville (MIT Press) is a comprehensive introduction into the subject.[2]

Scaling to Larger Datasets

In this book, we always assumed that the data we were working with could be stored in a NumPy array or SciPy sparse matrix in memory (RAM). Even though modern servers often have hundreds of gigabytes (GB) of RAM, this is a fundamental restriction on the size of data you can work with. Not everybody can afford to buy such a large machine, or even to rent one from a cloud provider. In most applications, the data that is used to build a machine learning system is relatively small, though, and few machine learning datasets consist of hundreds of gigabites of data or more. This makes expanding your RAM or renting a machine from a cloud provider a viable solution in many cases. If you need to work with terabytes of data, however, or you need

2 A preprint of *Deep Learning* can be viewed at *http://www.deeplearningbook.org/*.