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 While the progress in this field is so fast-paced that any current reference to the state of the art will soon be outdated, the recent book *Deep Learning* by Ian Goodfellow, Yoshua Bengio, and Aaron Courville (MIT Press) is a comprehensive introduction into the subject.[2]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/*.

to process large amounts of data on a budget, there are two basic strategies: *out-of-core learning* and *parallelization over a cluster*.

Out-of-core learning describes learning from data that cannot be stored in main memory, but where the learning takes place on a single computer (or even a single processor within a computer). The data is read from a source like the hard disk or the network either one sample at a time or in chunks of multiple samples, so that each chunk fits into RAM. This subset of the data is then processed and the model is updated to reflect what was learned from the data. Then, this chunk of the data is discarded and the next bit of data is read. Out-of-core learning is implemented for some of the models in `scikit-learn`, and you can find details on it in the online user guide. Because out-of-core learning requires all of the data to be processed by a single computer, this can lead to long runtimes on very large datasets. Also, not all machine learning algorithms can be implemented in this way.

The other strategy for scaling is distributing the data over multiple machines in a compute cluster, and letting each computer process part of the data. This can be much faster for some models, and the size of the data that can be processed is only limited by the size of the cluster. However, such computations often require relatively complex infrastructure. One of the most popular distributed computing platforms at the moment is the `spark` platform built on top of Hadoop. `spark` includes some machine learning functionality within the `MLLib` package. If your data is already on a Hadoop filesystem, or you are already using `spark` to preprocess your data, this might be the easiest option. If you don't already have such infrastructure in place, establishing and integrating a `spark` cluster might be too large an effort, however. The `vw` package mentioned earlier provides some distributed features and might be a better solution in this case.

Honing Your Skills

As with many things in life, only practice will allow you to become an expert in the topics we covered in this book. Feature extraction, preprocessing, visualization, and model building can vary widely between different tasks and different datasets. Maybe you are lucky enough to already have access to a variety of datasets and tasks. If you don't already have a task in mind, a good place to start is machine learning competitions, in which a dataset with a given task is published, and teams compete in creating the best possible predictions. Many companies, nonprofit organizations, and universities host these competitions. One of the most popular places to find them is Kaggle, a website that regularly holds data science competitions, some of which have substantial prize money attached.

The Kaggle forums are also a good source of information about the latest tools and tricks in machine learning, and a wide range of datasets are available on the site. Even more datasets with associated tasks can be found on the OpenML platform, which