PREV
7. Working with Text Data

NEXT
Index

Chapter 8. Wrapping Up

You now know how to apply the important machine learning algorithms for supervised and unsupervised learning, which allow you to solve a wide variety of machine learning problems. Before we leave you to explore all the possibilities that machine learning offers, we want to give you some final words of advice, point you toward some additional resources, and give you suggestions on how you can further improve your machine learning and data science skills.

8.1 Approaching a Machine Learning Problem

With all the great methods that we introduced in this book now at your fingertips, it may be tempting to jump in and start solving your data-related problem by just running your favorite algorithm. However, this is not usually a good way to begin your analysis. The machine learning algorithm is usually only a small part of a larger data analysis and decision-making process. To make effective use of machine learning, we need to take a step back and consider the problem at large. First, you should think about what kind of question you want to answer. Do you want to do exploratory analysis and just see if you find something interesting in the data? Or do you already have a particular goal in mind? Often you will start with a goal, like detecting fraudulent user transactions, making movie recommendations, or finding unknown planets. If you have such a goal, before building a system to achieve it, you should first think about how to define and measure success, and what the impact of a successful solution would be to your overall business or research goals. Let's say your goal is fraud detection.

Then the following questions open up:

- How do I measure if my fraud prediction is actually working?
- Do I have the right data to evaluate an algorithm?
- If I am successful, what will be the business impact of my solution?

As we discussed in Chapter 5, it is best if you can measure the performance of your algorithm directly using a business metric, like increased profit or decreased losses. This is often hard to do, though. A question that can be easier to answer is "What if I built the perfect model?" If perfectly detecting any fraud will save your company $100 a month, these possible savings will probably not be enough

hand, if the model might save your company tens of thousands of dollars every month, the problem might be worth exploring.

Say you've defined the problem to solve, you know a solution might have a significant impact for your project, and you've ensured that you have the right information to evaluate success. The next steps are usually acquiring the data and building a working prototype. In this book we have talked about many models you can employ, and how to properly evaluate and tune these models. While trying out models, though, keep in mind that this is only a small part of a larger data science workflow, and model building is often part of a feedback circle of collecting new data, cleaning data, building models, and analyzing the models. Analyzing the mistakes a model makes can often be informative about what is missing in the data, what additional data could be collected, or how the task could be reformulated to make machine learning more effective. Collecting more or different data or changing the task formulation slightly might provide a much higher payoff than running endless grid searches to tune parameters.

8.1.1 Humans in the Loop

You should also consider if and how you should have humans in the loop. Some processes (like pedestrian detection in a self-driving car) need to make immediate decisions. Others might not need immediate responses, and so it can be possible to have humans confirm uncertain decisions. Medical applications, for example, might need very high levels of precision that possibly cannot be achieved by a machine learning algorithm alone. But if an algorithm can make 90 percent, 50 percent, or maybe even just 10 percent of decisions automatically, that might already decrease response time or reduce cost. Many applications are dominated by "simple cases," for which an algorithm can make a decision, with relatively few "complicated cases," which can be rerouted to a human.

8.2 From Prototype to Production

The tools we've discussed in this book are great for many machine learning applications, and allow very quick analysis and prototyping. Python and `scikit-learn` are also used in production systems in many organizations—even very large ones like international banks and global social media companies. However, many companies have complex infrastructure, and it is not always easy to include Python in these systems. That is not necessarily a problem. In many companies, the data analytics teams work with languages like Python and R that allow the quick testing of ideas, while production teams work with languages like Go, Scala, C++, and Java to build robust, scalable systems. Data analysis has different requirements from building live services, and so using different languages for these tasks makes sense. A relatively common solution is to reimplement the solution that was found by the analytics team inside the larger framework, using a high-performance language. This can be easier than embedding a whole library or programming language and converting from and to the different data formats.

Regardless of whether you can use `scikit-learn` in a production system or not, it is important to keep in mind that production systems have different requirements from one-off analysis scripts. If an algorithm is deployed into a larger system, software engineering aspects like reliability, predictability, runtime, and memory requirements gain relevance. Simplicity is key in providing machine learning systems that perform well in these areas. Critically inspect each part of your data processing and prediction pipeline and ask yourself how much complexity each step creates, how robust each component is to changes in the data or compute infrastructure, and if the benefit of each component warrants the complexity. If you are building involved machine learning systems, we highly recommend reading the paper "Machine Learning: The High Interest Credit Card of Technical Debt" (http://research.google.com/pubs/pub43146.html), published by researchers in Google's machine learning team. The paper highlights the trade-off in creating and maintaining machine learning software in production at a large

8.3 Testing Production Systems

In this book, we covered how to evaluate algorithmic predictions based on a test set that we collected beforehand. This is known as *offline evaluation*. If your machine learning system is user-facing, this is only the first step in evaluating an algorithm, though. The next step is usually *online testing* or *live testing*, where the consequences of employing the algorithm in the overall system are evaluated. Changing the recommendations or search results users are shown by a website can drastically change their behavior and lead to unexpected consequences. To protect against these surprises, most user-facing services employ *A/B testing*, a form of blind user study. In A/B testing, without their knowledge a selected portion of users will be provided with a website or service using algorithm A, while the rest of the users will be provided with algorithm B. For both groups, relevant success metrics will be recorded for a set period of time. Then, the metrics of algorithm A and algorithm B will be compared, and a selection between the two approaches will be made according to these metrics. Using A/B testing enables us to evaluate the algorithms "in the wild," which might help us to discover unexpected consequences when users are interacting with our model. Often A is a new model, while B is the established system. There are more elaborate mechanisms for online testing that go beyond A/B testing, such as *bandit algorithms*. A great introduction to this subject can be found in the book *Bandit Algorithms for Website Optimization* (http://shop.oreilly.com/product/0636920027393.do) by John Myles White (O'Reilly).

8.4 Building Your Own Estimator

This book has covered a variety of tools and algorithms implemented in `scikit-learn` that can be used on a wide range of tasks. However, often there will be some particular processing you need to do for your data that is not implemented in `scikit-learn`. It may be enough to just preprocess your data before passing it to your `scikit-learn` model or pipeline. However, if your preprocessing is data dependent, and you want to apply a grid search or cross-validation, things become trickier.

In Chapter 6 we discussed the importance of putting all data-dependent processing inside the cross-validation loop. So how can you use your own processing together with the `scikit-learn` tools? There is a simple solution: build your own estimator! Implementing an estimator that is compatible with the `scikit-learn` interface, so that it can be used with `Pipeline`, `GridSearchCV`, and `cross_val_score`, is quite easy. You can find detailed instructions in the `scikit-learn` documentation (http://scikit-learn.org/stable/developers/contributing.html#rolling-your-own-estimator), but here is the gist. The simplest way to implement a transformer class is by inheriting from `BaseEstimator` and `TransformerMixin`, and then implementing the `__init__`, `fit`, and `predict` functions like this:

In[1]:

```
from sklearn.base import BaseEstimator, TransformerMixin

class MyTransformer(BaseEstimator, TransformerMixin):
    def __init__(self, first_parameter=1, second_parameter=2):
        # All parameters must be specified in the __init__ function
        self.first_parameter = 1
        self.second_parameter = 2

    def fit(self, X, y=None):
        # fit should only take X and y as parameters
        # Even if your model is unsupervised, you need to accept a y a

        # Model fitting code goes here
        print("fitting the model right here")
        # fit returns self
        return self

    def transform(self, X):
```

```
        X_transformed = X + 1
        return X_transformed
```

Implementing a classifier or regressor works similarly, only instead of `TransformerMixin` you need to inherit from `ClassifierMixin` or `RegressorMixin`. Also, instead of implementing `transform`, you would implement `predict`.

As you can see from the example given here, implementing your own estimator requires very little code, and most `scikit-learn` users build up a collection of custom models over time.

8.5 Where to Go from Here

This book provides an introduction to machine learning and will make you an effective practitioner. However, if you want to further your machine learning skills, here are some suggestions of books and more specialized resources to investigate to dive deeper.

8.5.1 Theory

In this book, we tried to provide an intuition of how the most common machine learning algorithms work, without requiring a strong foundation in mathematics or computer science. However, many of the models we discussed use principles from probability theory, linear algebra, and optimization. While it is not necessary to understand all the details of how these algorithms are implemented, we think that knowing some of the theory behind the algorithms will make you a better data scientist. There have been many good books written about the theory of machine learning, and if we were able to excite you about the possibilities that machine learning opens up, we suggest you pick up at least one of them and dig deeper. We already mentioned Hastie, Tibshirani, and Friedman's book *The Elements of Statistical Learning* in the Preface, but it is worth repeating this recommendation here. Another quite accessible book, with accompanying Python code, is *Machine Learning: An Algorithmic Perspective* by Stephen Marsland (Chapman and Hall/CRC). Two other highly recommended classics are *Pattern Recognition and Machine Learning* by Christopher Bishop (Springer), a book that emphasizes a probabilistic framework, and *Machine Learning: A Probabilistic Perspective* by Kevin Murphy (MIT Press), a comprehensive (read: 1,000+ pages) dissertation on machine learning methods featuring in-depth discussions of state-of-the-art approaches, far beyond what we could cover in this book.

8.5.2 Other Machine Learning Frameworks and Packages

While `scikit-learn` is our favorite package for machine learning[1] and Python is our favorite language for machine learning, there are many other options out there. Depending on your needs, Python and `scikit-learn` might not be the best fit for your particular situation. Often using Python is great for trying out and evaluating models, but larger web services and applications are more commonly written in Java or C++, and integrating into these systems might be necessary for your model to be deployed. Another reason you might want to look beyond `scikit-learn` is if you are more interested in statistical modeling and inference than prediction. In this case, you should consider the `statsmodels` package for Python, which implements several linear models with a more statistically minded interface. If you are not married to Python, you might also consider using R, another lingua franca of data scientists. R is a language designed specifically for statistical analysis and is famous for its excellent visualization capabilities and the availability of many (often highly specialized) statistical modeling packages.

Another popular machine learning package is `xgboost` (often called xg-

datasets and for streaming data. For running machine learning algorithms distributed on a cluster, one of the most popular solutions at the time of writing is `mllib`, a Scala library built on top of the `spark` distributed computing environment.

8.5.3 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" (http://www.netflixprize.com/), 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.

8.5.4 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

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.

8.5.5 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]

8.5.6 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 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 (http://scikit-learn.org/stable/modules/scaling_strategies.html#scaling-with-instances-using-out-of-core-learning). 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 interacting a `spark` cluster might

8.5.7 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 (http://www.openml.org/), which hosts over 20,000 datasets with over 50,000 associated machine learning tasks. Working with these datasets can provide a great opportunity to practice your machine learning skills. A disadvantage of competitions is that they already provide a particular metric to optimize, and usually a fixed, preprocessed dataset. Keep in mind that defining the problem and collecting the data are also important aspects of real-world problems, and that representing the problem in the right way might be much more important than squeezing the last percent of accuracy out of a classifier.

8.6 Conclusion

We hope we have convinced you of the usefulness of machine learning in a wide variety of applications, and how easily machine learning can be implemented in practice. Keep digging into the data, and don't lose sight of the larger picture.

---

1 Andreas might not be entirely objective in this matter.

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