Machine Learning in Python®: Essential Techniques for Predictive Analysis
By Michael Bowles

CHAPTER
1

The Two Essential Algorithms for Making Predictions

This book focuses on the machine learning process and so covers just a few of the most effective and widely used algorithms. It does not provide a survey of machine learning techniques. Too many of the algorithms that might be included in a survey are not actively used by practitioners.

This book deals with one class of machine learning problems, generally referred to as *function approximation*. Function approximation is a subset of problems that are called *supervised learning* problems. Linear regression and its classifier cousin, logistic regression, provide familiar examples of algorithms for function approximation problems. Function approximation problems include an enormous breadth of practical classification and regression problems in all sorts of arenas, including text classification, search responses, ad placements, spam filtering, predicting customer behavior, diagnostics, and so forth. The list is almost endless.

Broadly speaking, this book covers two classes of algorithms for solving function approximation problems: penalized linear regression methods and ensemble methods. This chapter introduces you to both of these algorithms, outlines some of their characteristics, and reviews the results of comparative studies of algorithm performance in order to demonstrate their consistent high performance.

This chapter then discusses the process of building predictive models. It describes the kinds of problems that you'll be able to address with the tools covered here and the flexibilities that you have in how you set up your problem and define the features that you'll use for making predictions. It describes process steps involved in building a predictive model and qualifying it for deployment.

Why Are These Two Algorithms So Useful?

Several factors make the penalized linear regression and ensemble methods a useful collection. Stated simply, they will provide optimum or near-optimum performance on the vast majority of predictive analytics (function approximation) problems encountered in practice, including big data sets, little data sets, wide data sets, tall skinny data sets, complicated problems, and simple problems. Evidence for this assertion can be found in two papers by Rich Caruana and his colleagues:

- "An Empirical Comparison of Supervised Learning Algorithms," by Rich Caruana and Alexandru Niculescu-Mizil[1]
- "An Empirical Evaluation of Supervised Learning in High Dimensions," by Rich Caruana, Nikos Karampatziakis, and Ainur Yessenalina[2]

In those two papers, the authors chose a variety of classification problems and applied a variety of different algorithms to build predictive models. The models were run on test data that were not included in training the models, and then the algorithms included in the studies were ranked on the basis of their performance on the problems. The first study compared 9 different basic algorithms on 11 different machine learning (binary classification) problems. The problems used in the study came from a wide variety of areas, including demographic data, text processing, pattern recognition, physics, and biology. Table 1-1 lists the data sets used in the study using the same names given by the study authors. The table shows how many attributes were available for predicting outcomes for each of the data sets, and it shows what percentage of the examples were positive.

The term *positive example* in a classification problem means an experiment (a line of data from the input data set) in which the outcome is positive. For example, if the classifier is being designed to determine whether a radar return signal indicates the presence of an airplane, then the positive example would be those returns where there was actually an airplane in the radar's field of view. The term *positive* comes from this sort of example where the two outcomes represent presence or absence. Other examples include presence or absence of disease in a medical test or presence or absence of cheating on a tax return.

Not all classification problems deal with presence or absence. For example, determining the gender of an author by machine-reading their text or

machine-analyzing a handwriting sample has two classes—male and female—but there's no sense in which one is the absence of the other. In these cases, there's some arbitrariness in the assignment of the designations "positive" and "negative." The assignments of positive and negative can be arbitrary, but once chosen must be used consistently.

Some of the problems in the first study had many more examples of one class than the other. These are called *unbalanced*. For example, the two data sets Letter.p1 and Letter.p2 pose closely related problems in correctly classifying typed uppercase letters in a wide variety of fonts. The task with Letter.p1 is to correctly classify the letter *O* in a standard mix of letters. The task with Letter.p2 is to correctly classify *A–M* versus *N–Z*. The percentage of positives shown in Table 1-1 reflects this difference.

Table 1-1 also shows the number of "attributes" in each of the data sets. Attributes are the variables you have available to base a prediction on. For example, to predict whether an airplane will arrive at its destination on time or not, you might incorporate attributes such as the name of the airline company, the make and year of the airplane, the level of precipitation at the destination airport, the wind speed and direction along the flight path, and so on. Having a lot of attributes upon which to base a prediction can be a blessing and a curse. Attributes that relate directly to the outcomes being predicted are a blessing. Attributes that are unrelated to the outcomes are a curse. Telling the difference between blessed and cursed attributes requires data. Chapter 3, "Predictive Model Building: Balancing Performance, Complexity, and Big Data," goes into that in more detail.

Table 1-1: Sketch of Problems in Machine Learning Comparison Study[3]

DATA SET NAME	NUMBER OF ATTRIBUTES	% OF EXAMPLES THAT ARE POSITIVE
Adult	14	25
Bact	11	69
Cod	15	50
Calhous	9	52
Cov_Type	54	36
HS	200	24
Letter.p1	16	3
Letter.p2	16	53
Medis	63	11
Mg	124	17
Slac	59	50

Table 1-2 shows how the algorithms covered in this book fared relative to the other algorithms used in the study. Table 1-2 shows which algorithms showed the top five performance scores for each of the problems listed in Table 1-1. Algorithms covered in this book are spelled out (boosted decision trees, Random Forests, bagged decision trees, and logistic regression). The first three of these are ensemble methods. Penalized regression was not fully developed when the study was done and wasn't evaluated. Logistic regression is a close relative and is used to gauge the success of regression methods. Each of the 9 algorithms used in the study had 3 different data reduction techniques applied, for a total of 27 combinations. The top five positions represent roughly the top 20 percent of performance scores. The row next to the heading Covt indicates that the boosted decision trees algorithm was the first and second best relative to performance, Random Forests algorithm was the fourth and fifth best, and bagged decision trees algorithm was the third best. In the cases where algorithms not covered here were in the top five, an entry appears in the Other column. The algorithms that show up there are *k nearest neighbors* (KNNs), *artificial neural nets* (ANNs), and *support vector machines* (SVMs).

Table 1-2: How the Algorithms Covered in This Book Compare on Different Problems

ALGORITHM	BOOSTED DECISION TREES	RANDOM FORESTS	BAGGED DECISION TREES	LOGISTIC REGRESSION	OTHER
Covt	1, 2	4, 5	3		
Adult	1, 4	2	3, 5		
LTR.P1	1				SVM, KNN
LTR.P2	1, 2	4, 5			SVM
MEDIS		1, 3		5	ANN
SLAC		1, 2, 3	4, 5		
HS	1, 3				ANN
MG		2, 4, 5	1, 3		
CALHOUS	1, 2	5	3, 4		
COD	1, 2		3, 4, 5		
BACT	2, 5		1, 3, 4		

Logistic regression captures top-five honors in only one case in Table 1-2. The reason for that is that these data sets have few attributes (at most 200) relative to examples (5,000 in each data set). There's plenty of data to resolve a model with so few attributes, and yet the training sets are small enough that the training time is not excessive.

NOTE **As you'll see in Chapter 3 and in the examples covered in Chapter 5, "Building Predictive Models Using Penalized Linear Methods," and Chapter 7, "Building Ensemble Models with Python," the penalized regression methods perform best relative to other algorithms when there are numerous attributes and not enough examples or time to train a more complicated ensemble model.**

Caruana et al. have run a newer study (2008) to address how these algorithms compare when the number of attributes increases. That is, how do these algorithms compare on big data? A number of fields have significantly more attributes than the data sets in the first study. For example, genomic problems have several tens of thousands of attributes (one attribute per gene), and text mining problems can have millions of attributes (one attribute per distinct word or per distinct pair of words). Table 1-3 shows how linear regression and ensemble methods fare as the number of attributes grows. The results in Table 1-3 show the ranking of the algorithms used in the second study. The table shows the performance on each of the problems individually and in the far right column shows the ranking of each algorithm's average score across all the problems. The algorithms used in the study are broken into two groups. The top group of algorithms are ones that will be covered in this book. The bottom group will not be covered.

The problems shown in Table 1-3 are arranged in order of their number of attributes, ranging from 761 to 685,569. Linear (logistic) regression is in the top three for 5 of the 11 test cases used in the study. Those superior scores were concentrated among the larger data sets. Notice that boosted decision tree (denoted by BSTDT in Table 1-3) and Random Forests (denoted by RF in Table 1-3) algorithms still perform near the top. They come in first and second for overall score on these problems.

The algorithms covered in this book have other advantages besides raw predictive performance. An important benefit of the penalized linear regression models that the book covers is the speed at which they train. On big problems, training speed can become an issue. In some problems, model training can take days or weeks. This time frame can be an intolerable delay, particularly early in development when iterations are required to home in on the best approach. Besides training very quickly, after being deployed a trained linear model can produce predictions very quickly—quickly enough for high-speed trading or Internet ad insertions. The study demonstrates that penalized linear regression can provide the best answers available in many cases and be near the top even in cases where they are not the best.

In addition, these algorithms are reasonably easy to use. They do not have very many tunable parameters. They have well-defined and well-structured input types. They solve several types of problems in regression and classification. It is not unusual to be able to arrange the input data and generate a first trained model and performance predictions within an hour or two of starting a new problem.

Table 1-3: How the Algorithms Covered in This Book Compare on Big Data Problems

DIM	761 STURN	761 CALAM	780 DIGITS	927 TIS	1344 CRYST	3448 KDD98	20958 R-S	105354 CITE	195203 DSE	405333 SPAM	685569 IMDB	MEAN
BSTDT	8	1	2	6	1	3	8	1	7	6	3	1
RF	9	4	3	3	2	1	6	5	3	1	3	2
BAGDT	5	2	6	4	3	1	9	1	6	7	3	4
BSTST	2	3	7	7	7	1	7	4	8	8	5	7
LR	4	8	9	1	4	1	2	2	2	4	4	6
SVM	3	5	5	2	5	2	1	1	5	5	3	3
ANN	6	7	4	5	8	1	4	2	1	3	3	5
KNN	1	6	1	9	6	2	10	1	7	9	6	8
PRC	7	9	8	8	7	1	3	3	4	2	2	9
NB	10	10	10	10	9	1	5	1	9	10	7	10

One of their most important features is that they indicate which of their input variables is most important for producing predictions. This turns out to be an invaluable feature in a machine learning algorithm. One of the most time-consuming steps in the development of a predictive model is what is sometimes called *feature selection* or *feature engineering*. This is the process whereby the data scientist chooses the variables that will be used to predict outcomes. By ranking features according to importance, the algorithms covered in this book aid in the feature-engineering process by taking some of the guesswork out of the development process and making the process more sure.

What Are Penalized Regression Methods?

Penalized linear regression is a derivative of *ordinary least squares* (OLS) regression—a method developed by Gauss and Legendre roughly 200 years ago. Penalized linear regression methods were designed to overcome some basic limitations of OLS regression. The basic problem with OLS is that sometimes it overfits the problem. Think of OLS as fitting a line through a group of points, as in Figure 1-1. This is a simple prediction problem: predicting y, the target value given a single attribute x. For example, the problem might be to predict men's salaries using only their heights. Height is slightly predictive of salaries for men (but not for women).

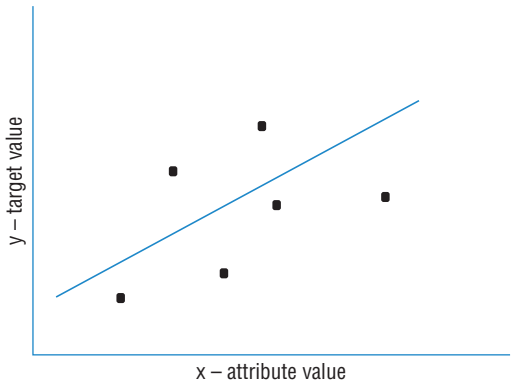

Figure 1-1: Ordinary least squares fit

The points represent men's salaries versus their heights. The line in Figure 1-1 represents the OLS solution to this prediction problem. In some sense, the line is the best predictive model for men's salaries given their heights. The data set has six points in it. Suppose that the data set had only two points in it. Imagine that there's a population of points, like the ones in Figure 1-1, but that you do not get to see all the points. Maybe they are too expensive to generate, like the

genetic data mentioned earlier. There are enough humans available to isolate the gene that is the culprit; the problem is that you do not have gene sequences for many of them because of cost.

To simulate this in the simple example, imagine that instead of six points you're given only two of the six points. How would that change the nature of the line fit to those points? It would depend on which two points you happened to get. To see how much effect that would have, pick any two points from Figure 1-1 and imagine a line through them. Figure 1-2 shows some of the possible lines through pairs of points from Figure 1-1. Notice how much the lines vary depending on the choice of points.

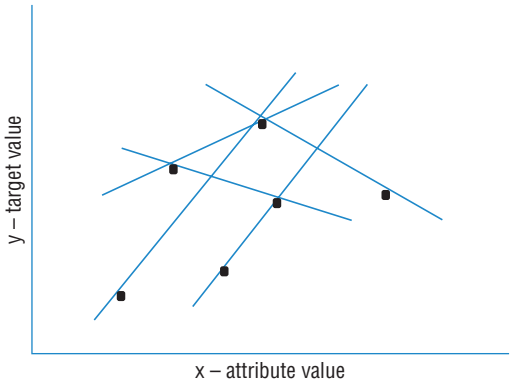

Figure 1-2: Fitting lines with only two points

The problem with having only two points to fit a line is that there is not enough data for the number of degrees of freedom. A line has two degrees of freedom. Having two degrees of freedom means that there are two independent parameters that uniquely determine a line. You can imagine grabbing hold of a line in the plane and sliding it up and down in the plane or twisting it to change its slope. So, vertical position and slope are independent. They can be changed separately, and together they completely specify a line. The degrees of freedom of a line can be expressed in several equivalent ways (where it intercepts the y-axis and its slope, two points that are on the line, and so on). All of these representations of a line require two parameters to specify.

When the number of degrees of freedom is equal to the number of points, the predictions are not very good. The lines hit the points used to draw them, but there is a lot of variation among lines drawn with different pairs of points. You cannot place much faith in a prediction that has as many degrees of freedom as the number of points in your data set. The plot in Figure 1-1 had six points and fit a line (two degrees of freedom) through them. That is six points and two degrees of freedom. The thought problem of determining the genes causing a heritable condition illustrated that having more genes to choose from makes it

necessary to have more data in order to isolate a cause from among the 20,000 or so possible human genes. The 20,000 different genes represent 20,000 degrees of freedom. Data from even 20,000 different persons will not suffice to get a reliable answer, and in many cases, all that can be afforded within the scope of a reasonable study is a sample from 500 or so persons. That is where penalized linear regression may be the best algorithm choice.

Penalized linear regression provides a way to systematically reduce degrees of freedom to match the amount of data available and the complexity of the underlying phenomena. These methods have become very popular for problems with very many degrees of freedom. They are a favorite for genetic problems where the number of degrees of freedom (that is, the number of genes) can be several tens of thousands and for problems like text classification where the number of degrees of freedom can be more than a million. Chapter 4, "Penalized Linear Regression," gives more detail on how these methods work, sample code that illustrates the mechanics of these algorithms, and examples of the process for implementing machine learning systems using available Python packages.

What Are Ensemble Methods?

The other family of algorithms covered in this book is ensemble methods. The basic idea with ensemble methods is to build a horde of different predictive models and then combine their outputs—by averaging the outputs or taking the majority answer (voting). The individual models are called *base learners*. Some results from computational learning theory show that if the base learners are just slightly better than random guessing, the performance of the ensemble can be very good if there is a sufficient number of independent models.

One of the problems spurring the development of ensemble methods has been the observation that some particular machine learning algorithms exhibit instability. For example, the addition of fresh data to the data set might result in a radical change in the resulting model or its performance. Binary decision trees and traditional neural nets exhibit this sort of instability. This instability causes high variance in the performance of models, and averaging many models can be viewed as a way to reduce the variance. The trick is how to generate large numbers of independent models, particularly if they are all using the same base learner. Chapter 6, "Ensemble Methods," will get into the details of how this is done. The techniques are ingenious, and it is relatively easy to understand their basic principles of operation. Here is a preview of what's in store.

The ensemble methods that enjoy the widest availability and usage incorporate binary decision trees as their base learners. Binary decision trees are often portrayed as shown in Figure 1-3. The tree in Figure 1-3 takes a real number, called x, as input at the top, and then uses a series of binary (two-valued) decisions to decide what value should be output in response to x. The first decision

is whether x is less than 5. If the answer to that question is "no," the binary decision tree outputs the value 4 indicated in the circle below the No leg of the upper decision box. Every possible value for x leads to some output y from the tree. Figure 1-4 plots the output (y) as a function of the input to the tree (x).

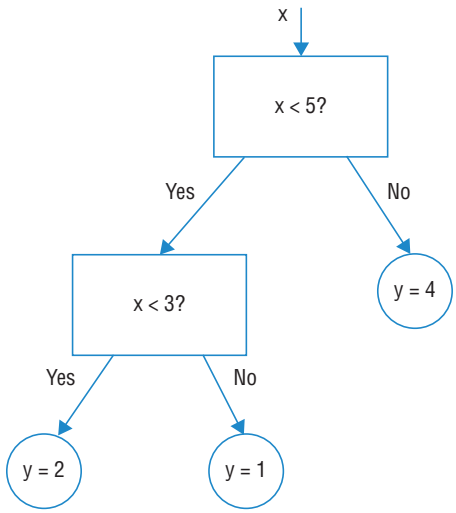

Figure 1-3: Binary decision tree example

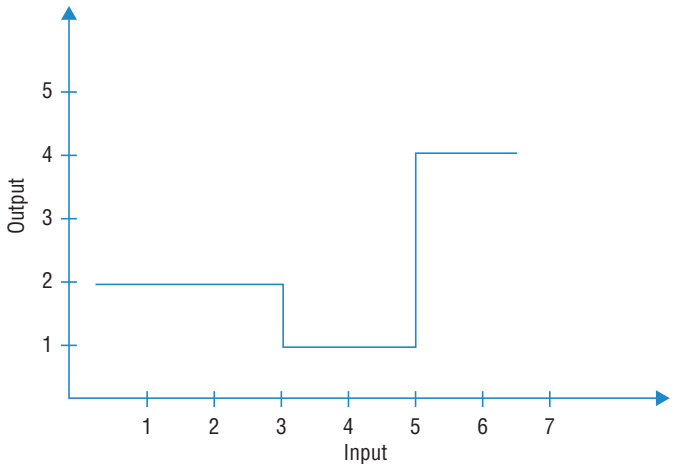

Figure 1-4: Input-output graph for the binary decision tree example

This description raises the question of where the comparisons (for example, $x < 5$?) come from and where the output values (in the circles at the bottom of the tree) come from. These values come from training the binary tree on the input data. The algorithm for doing that training is not difficult to understand and is

covered in Chapter 6. The important thing to note at this point is that the values in the trained binary decision tree are fixed, given the data. The process for generating the tree is deterministic. One way to get differing models is to take random samples of the training data and train on these random subsets. That technique is called *Bagging* (short for *bootstrap aggregating*). It gives a way to generate a large number of slightly different binary decision trees. Those are then averaged (or voted for a classifier) to yield a final result. Chapter 6 describes in more detail this technique and other more powerful ones.

How to Decide Which Algorithm to Use

Table 1-4 gives a sketch comparison of these two families of algorithms. Penalized linear regression methods have the advantage that they train very quickly. Training times on large data sets can extend to hours, days, or even weeks. Training usually needs to be done several times before a deployable solution is arrived at. Long training times can stall development and deployment on large problems. The rapid training time for penalized linear methods makes them useful for the obvious reason that shorter is better. Depending on the problem, these methods may suffer some performance disadvantages relative to ensemble methods. Chapter 3 gives more insight into the types of problems where penalized regression might be a better choice and those where ensemble methods might be a better choice. Penalized linear methods can sometimes be a useful first step in your development process even in the circumstance where they yield inferior performance to ensemble methods.

Early in development, a number of training iterations will be necessary for purposes of feature selection and feature engineering and for solidifying the mathematical problem statement. Deciding what you are going to use as input to your predictive model can take some time and thought. Sometimes that is obvious, but usually it requires some iteration. Throwing in everything you can find is not usually a good solution.

Trial and error is typically required to determine the best inputs for a model. For example, if you're trying to predict whether a visitor to your website will click a link for an ad, you might try using demographic data for the visitor. Maybe that does not give you the accuracy that you need, so you try incorporating data regarding the visitor's past behavior on the site—what ad the visitor clicked during past site visits or what products the visitor has bought. Maybe adding data about the site the visitor was on before coming to your site would help. These questions lead to a series of experiments where you incorporate the new data and see whether it hurts or helps. This iteration is generally time-consuming both for the data manipulations and for training your predictive model. Penalized linear regression will generally be faster than an ensemble method, and the time difference can be a material factor in the development process.

For example, if the training set is on the order of a gigabyte, training times may be on the order of 30 minutes for penalized linear regression and 5 or 6 hours for an ensemble method. If the feature engineering process requires 10 iterations to select the best feature set, the computation time alone comes to the difference between taking a day or taking a week to accomplish feature engineering. A useful process, therefore, is to train a penalized linear model in the early stages of development, feature engineering, and so on. That gives the data scientist a feel for which variables are going to be useful and important as well as a baseline performance for comparison with other algorithms later in development.

Besides enjoying a training time advantage, penalized linear methods generate predictions much faster than ensemble methods. Generating a prediction involves using the trained model. The trained model for penalized linear regression is simply a list of real numbers—one for each feature being used to make the predictions. The number of floating-point operations involved is the number of variables being used to make predictions. For highly time-sensitive predictions such as high-speed trading or Internet ad insertions, computation time makes the difference between making money and losing money.

Table 1-4: High-Level Tradeoff between Penalized Linear Regression and Ensemble Algorithms

	TRAINING SPEED	PREDICTION SPEED	PROBLEM COMPLEXITY	DEALS WITH WIDE ATTRIBUTE
Penalized Linear Regression	+	+	–	+
Ensemble Methods	–	–	+	–

For some problems, linear methods may give equivalent or even better performance than ensemble methods. Some problems do not require complicated models. Chapter 3 goes into some detail about the nature of problem complexity and how the data scientist's task is to balance problem complexity, predictive model complexity, and data set size to achieve the best deployable model. The basic idea is that on problems that are not complex and problems for which sufficient data are not available, linear methods may achieve better overall performance than more complicated ensemble methods. Genetic data provide a good illustration of this type of problem.

The general perception is that there's an enormous amount of genetic data around. Genetic data sets are indeed large when measured in bytes, but in terms of generating accurate predictions, they aren't very large. To understand this distinction, consider the following thought experiment. Suppose that you have two people, one with a heritable condition and the other without. If you

had genetic sequences for the two people, could you determine which gene was responsible for the condition? Obviously, that's not possible because many genes will differ between the two persons. So how many people would it take? At a minimum, it would take gene sequences for as many people as there are genes, and given any noise in the measurements, it would take even more. Humans have roughly 20,000 genes, depending on your count. And each datum costs roughly $1,000. So having just enough data to resolve the disease with perfect measurements would cost $20 million.

This situation is very similar to fitting a line to two points, as discussed earlier in this chapter. Models need to have fewer degrees of freedom than the number of data points. The data set typically needs to be a multiple of the degrees of freedom in the model. Because the data set size is fixed, the degrees of freedom in the model need to be adjustable. The chapters dealing with penalized linear regression will show you how the adjustability is built into penalized linear regression and how to use it to achieve optimum performance.

NOTE **The two broad categories of algorithms addressed in this book match those that Jeremy Howard and I presented at Strata Conference in 2012. Jeremy took ensemble methods, and I took penalized linear regression. We had fun arguing about the relative merits of the two groups. In reality, however, those two cover something like 80 percent of the model building that I do, and there are good reasons for that.**

Chapter 3 goes into more detail about why one algorithm or another is a better choice for a given problem. It has to do with the complexity of the problem and the number of degrees of freedom inherent in the algorithms. The linear models tend to train rapidly and often give equivalent performance to nonlinear ensemble methods, especially if the data available are somewhat constrained. Because they're so rapid to train, it is often convenient to train linear models for early feature selection and to ballpark achievable performance for a specific problem. The linear models considered in this book can give information about variable importance to aid in the feature selection process. The ensemble methods often give better performance if there are adequate data and also give somewhat indirect measures of relative variable importance.

The Process Steps for Building a Predictive Model

Using machine learning requires several different skills. One is the required programming skill, which this book does not address. The other skills have to do with getting an appropriate model trained and deployed. These other skills are what the book does address. What do these other skills include?

Initially, problems are stated in somewhat vague language-based terms like "Show site visitors links that they're likely to click on." To turn this into a working system requires restating the problem in concrete mathematical terms, finding data to base the prediction on, and then training a predictive model that will predict the likelihood of site visitors clicking the links that are available for presentation. Stating the problem in mathematical terms makes assumptions about what features will be extracted from the available data sources and how they will be structured.

How do you get started with a new problem? First, you look through the available data to determine which of the data might be of use in prediction. "Looking through the data" means running various statistical tests on the data to get a feel for what they reveal and how they relate to what you're trying to predict. Intuition can guide you to some extent. You can also quantify the outcomes and test the degree to which potential prediction features correlate with these outcomes. Chapter 2, "Understand the Problem by Understanding the Data," goes through this process for the data sets that are used to characterize and compare the algorithms outlined in the rest of the book.

By some means, you develop a set of features and start training the machine learning algorithm that you have selected. That produces a trained model and estimates its performance. Next, you want to consider making changes to the features set, including adding new ones or removing some that proved unhelpful, or perhaps changing to a different type of training objective (also called a *target*) to see whether it improves performance. You'll iterate various design decisions to determine whether there's a possibility of improving performance. You may pull out the examples that show the worst performance and then attempt to determine if there's something that unites these examples. That may lead to another feature to add to the prediction process, or it might cause you to bifurcate the data and train different models on different populations.

The goal of this book is to make these processes familiar enough to you that you can march through these development steps confidently. That requires your familiarity with the input data structures required by different algorithms as you frame the problem and begin extracting the data to be used in training and testing algorithms. The process usually includes several of the following steps:

1. Extract and assemble features to be used for prediction.
2. Develop targets for the training.
3. Train a model.
4. Assess performance on test data.

NOTE **The first pass can usually be improved on by trying different sets of features, different types of targets, and so on.**

Machine learning requires more than familiarization with a few packages. It requires understanding and having practiced the process involved in developing a deployable model. This book aims to give you that understanding. It assumes basic undergraduate math and some basic ideas from probability and statistics, but the book doesn't presuppose a background in machine learning. At the same time, it intends to arm readers with the very best algorithms for a wide class of problems, not necessarily to survey all machine learning algorithms or approaches. There are a number of algorithms that are interesting but that don't get used often, for a variety of reasons. For example, perhaps they don't scale well, maybe they don't give insight about what is going on inside, maybe they're difficult to use, and so on. It is well known, for example, that Random Forests (one of the algorithms covered here) is the leading winner of online machine competitions by a wide margin. There are good reasons why some algorithms are more often used by practitioners, and this book will succeed to the extent that you understand these when you've finished reading.

Framing a Machine Learning Problem

Beginning work on a machine learning competition presents a simulation of a real machine learning problem. The competition presents a brief description (for example, announcing that an insurance company would like to better predict loss rates on their automobile policies). As a competitor, your first step is to open the data set, take a look at the data available, and identify what form a prediction needs to take to be useful. The inspection of the data will give an intuitive feel for what the data represent and how they relate to the prediction job at hand. The data can give insight regarding approaches. Figure 1-5 depicts the process of starting from a general language statement of objective and moving toward an arrangement of data that will serve as input for a machine learning algorithm.

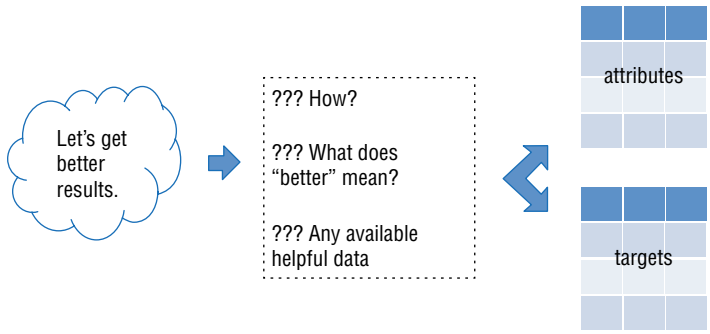

Figure 1-5: Framing a machine learning problem

The generalized statement caricatured as "Let's get better results" has first to be converted into specific goals that can be measured and optimized. For a website owner, specific performance might be improved click-through rates or more sales (or more contribution margin). The next step is to assemble data that might make it possible to predict how likely a given customer is to click various links or to purchase various products offered online. Figure 1-5 depicts these data as a matrix of attributes. For the website example, they might include other pages the visitor has viewed or items the visitor has purchased in the past. In addition to attributes that will be used to make predictions, the machine learning algorithms for this type of problem need to have correct answers to use for training. These are denoted as targets in Figure 1-5. The algorithms covered in this book learn by detecting patterns in past behaviors, but it is important that they not merely memorize past behavior; after all, a customer might not repeat a purchase of something he bought yesterday. Chapter 3 discusses in detail how this process of training without memorizing works.

Usually, several aspects of the problem formulation can be done in more than one way. This leads to some iteration between framing the problem, selecting and training a model, and producing performance estimates. Figure 1-6 depicts this process.

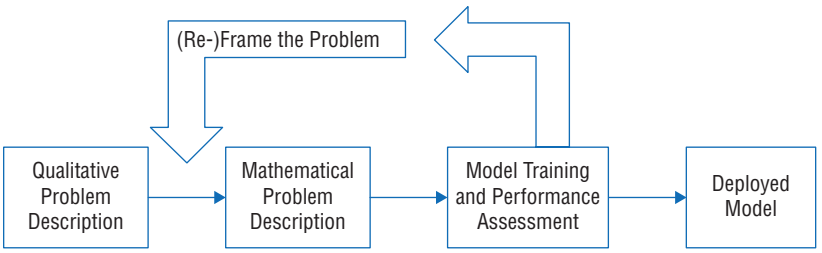

Figure 1-6: Iteration from formulation to performance

The problem may come with specific quantitative training objectives, or part of the job might be extracting these data (called *targets* or *labels*). Consider, for instance, the problem of building a system to automatically trade securities. To trade automatically, a first step might be to predict changes in the price of a security. The prices are easily available, so it is conceptually simple to use historical data to build training examples for which the future price changes are known. But even that involves choices and experimentation. Future price change could be computed in several different ways. The change could be the difference between the current price and the price 10 minutes in the future. It could also be the change between the current price and the price 10 days in the future. It could also be the difference between the current price and the maximum/minimum price over the next 10 minutes. The change in price could be characterized by a two-state variable taking values "higher" of "lower"

depending on whether the price is higher or lower 10 minutes in the future. Each of these choices will lead to a predictive model, and the predictions will be used for deciding whether to buy or sell the security. Some experimentation will be required to determine the best choice.

Feature Extraction and Feature Engineering

Deciding which variables to use for making predictions can also involve experimentation. This process is known as *feature extraction* and *feature engineering*. Feature extraction is the process of taking data from a free-form arrangement, such as words in a document or on a web page, and arranging them into rows and columns of numbers. For example, a spam-filtering problem begins with text from emails and might extract things such as the number of capital letters in the document and the number of words in all caps, the number of times the word "buy" appears in the document and other numeric features selected to highlight the differences between spam and non-spam emails.

Feature engineering is the process of manipulating and combining features to arrive at more informative ones. Building a system for trading securities involves feature extraction and feature engineering. Feature extraction would be deciding what things will be used to predict prices. Past prices, prices of related securities, interest rates, and features extracted from news releases have all been incorporated into various trading systems that have been discussed publicly. In addition, securities prices have a number of engineered features with names like stochastic, MACD (*moving average convergence divergence*), and RSI (*relative strength index*) that are basically functions of past prices that their inventors believed to be useful in securities trading.

After a reasonable set of features is developed, you can train a predictive model like the ones described in this book, assess its performance, and make a decision about deploying the model. Generally, you'll want to make changes to the features used, if for no other reason than to confirm that your model's performance is adequate. One way to determine which features to use is to try all combinations, but that can take a lot of time. Inevitably, you'll face competing pressures to improve performance but also to get a trained model into use quickly. The algorithms discussed in this book have the beneficial property of providing metrics on the utility of each attribute in producing predictions. One training pass will generate rankings on the features to indicate their relative importance. This information helps speed the feature engineering process.

NOTE **Data preparation and feature engineering is estimated to take 80 to 90 percent of the time required to develop a machine learning model.**

The model training process, which begins each time a baseline set of features is attempted, also involves a process. A modern machine learning algorithm,

such as the ones described in this book, trains something like 100 to 5,000 different models that have to be winnowed down to a single model for deployment. The reason for generating so many models is to provide models of all different shades of complexity. This makes it possible to choose the model that is best suited to the problem and data set. You don't want a model that's too simple or you give up performance, but you don't want a model that's too complicated or you'll overfit the problem. Having models in all shades of complexity lets you pick one that is just right.

Determining Performance of a Trained Model

The fit of a model is determined by how well it performs on data that were not used to train the model. This is an important step and conceptually simple. Just set aside some data. Don't use it in training. After the training is finished, use the data you set aside to determine the performance of your algorithm. This book discusses several systematic ways to hold out data. Different methods have different advantages, depending mostly on the size of the training data. As easy as it sounds, people continually figure out complicated ways to let the test data "leak" into the training process. At the end of the process, you'll have an algorithm that will sift through incoming data and make accurate predictions for you. It might need monitoring as changing conditions alter the underlying statistics.

Chapter Contents and Dependencies

Different readers may want to take different paths through this book, depending on their backgrounds and whether they have time to understand the basic principles. Figure 1-7 shows how chapters in the book depend on one another.

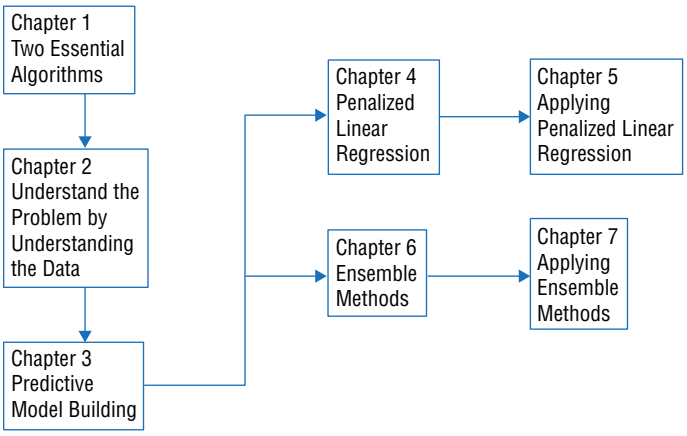

Figure 1-7: Dependence of chapters on one another

Chapter 2 goes through the various data sets that will be used for problem examples to illustrate the use of the algorithms that will be developed and to compare algorithms to each other based on performance and other features. The starting point with a new machine learning problem is digging into the data set to understand it better and to learn its problems and idiosyncrasies. Part of the point of Chapter 2 is to demonstrate some of the tools available in Python for data exploration. You might want to go through some but not all of the examples shown in Chapter 2 to become familiar with the process and then come back to Chapter 2 when diving into the solution examples later.

Chapter 3 explains the basic tradeoffs in a machine learning problem and introduces several key concepts that are used throughout the book. One key concept is the mathematical description of predictive problems. The basic distinctions between classification and regression problems are shown. Chapter 3 also introduces the concept of using out-of-sample data for determining the performance of a predictive model. Out-of-sample data are data that have not been included in the training of the model. Good machine learning practice demands that a developer produce solid estimates of how a predictive model will perform when it is deployed. This requires excluding some data from the training set and using it to simulate fresh data. The reasons for this requirement, the methods for accomplishing it, and the tradeoffs between different methods are described. Another key concept is that there are numerous measures of system performance. Chapter 3 outlines these methods and discusses tradeoffs between them. Readers who are already familiar with machine learning can browse this chapter and scan the code examples instead of reading it carefully and running the code.

Chapter 4 shows the core ideas of the algorithms for training penalized regression models. The chapter introduces the basic concepts and shows how the algorithms are derived. Some of the examples introduced in Chapter 3 are used to motivate the penalized linear regression methods and algorithms for their solution. The chapter runs through code for the core algorithms for solving penalized linear regression training. Chapter 4 also explains several extensions to linear regression methods. One of these extensions shows how to code factor variables as real numbers so that linear regression methods can be applied. Linear regression can be used only on problems where the predictors are real numbers; that is, the quantities being used to make predictions have to be numeric. Many practical and important problems have variables like "single, married, or divorced" that can be helpful in making predictions. To incorporate variables of this type (called *categorical variables*) in a linear regression model, means have been devised to convert categorical variables to real number variables. Chapter 4 covers those methods. In addition, Chapter 4 also shows methods (called *basis expansion*) for getting nonlinear functions out of nonlinear regression. Sometimes basis expansion can be used to squeeze a little more performance out of linear regression.

Chapter 5 applies the penalized regression algorithms developed in Chapter 4 to a number of the problems outlined in Chapter 2. The chapter outlines the Python packages that implement penalized regression methods and uses them to solve problems. The objective is to cover a wide enough variety of problems that practitioners can find a problem close to the one that they have in front of them to solve. Besides quantifying and comparing predictive performance, Chapter 5 looks at other properties of the trained algorithms. Variable selection and variable ranking are important to understand. This understanding will help speed development on new problems.

Chapter 6 develops ensemble methods. Because ensemble methods are most frequently based on binary decision trees, the first step is to understand the principles of training and using binary decision trees. Many of the properties of ensemble methods are ones that they inherit directly from binary decision trees. With that understanding in place, the chapter explains the three principal ensemble methods covered in the book. The common names for these are Bagging, boosting, and Random Forest. For each of these, the principles of operation are outlined and the code for the core algorithm is developed so that you can understand the principles of operation.

Chapter 7 uses ensemble methods to solve problems from Chapter 2 and then compares the various algorithms that have been developed. The comparison involves a number of elements. Predictive performance is one element of comparison. The time required for training and performance is another element. All the algorithms covered give variable importance ranking, and this information is compared on a given problem across several different algorithms.

In my experience, teaching machine learning to programmers and computer scientists, I've learned that code examples work better than mathematics for some people. The approach taken here is to provide some mathematics, algorithm sketches, and code examples to illustrate the important points. Nearly all the methods that are discussed will be found in the code included in the book and on the website. The intent is to provide hackable code to help you get up and running on your own problems as quickly as possible.

Summary

This chapter has given a specification for the kinds of problems that you'll be able to solve and a description of the process steps for building predictive models. The book concentrates on two algorithm families. Limiting the number of algorithms covered allows for a more thorough explanation of the background for these algorithms and of the mechanics of using them. This chapter showed some comparative performance results to motivate the choice of these two particular families. The chapter discussed the different strengths and characteristics of

these two families and gave some description of the types of problems that would favor one or the other of the two.

The chapter also laid out the steps in the process of developing a predictive model and elaborated on the tradeoffs and outcomes for each step. The use of data not included in model training was suggested for generating performance estimates for predictive models.

This book's goal is to bring programmers with little or no machine learning experience to the point where they feel competent and comfortable incorporating machine learning into projects. The book does not survey a wide number of algorithms. Instead, it covers several best-in-class algorithms that can offer you performance, flexibility, and clarity. Once you understand a little about how these work and have some experience using them, you'll find them easy and quick to use. They will enable you to solve a wide variety of problems without having to do a lot of fussing to get them trained, and they'll give you insight into the sources of their performance.

References

1. Caruana, Rich, and Alexandru Niculescu-Mizil. "An Empirical Comparison of Supervised Learning Algorithms." *Proceedings of the 23rd International Conference on Machine Learning.* ACM, 2006.
2. Caruana, Rich, Nikos Karampatziakis, and Ainur Yessenalina. "An Empirical Evaluation of Supervised Learning in High Dimensions." *Proceedings of the 25th International Conference on Machine Learning.* ACM, 2008.