Quote from the book The Master Algorithm

You could only buy books from your local bookstore, and your local bookstore had limited shelf space. But when you can download any book to your e-reader any time, the problem becomes the overwhelming number of choices. How do you browse the shelves of a bookstore that has millions of titles for sale? The same goes for other information goods: videos, music, news, tweets, blogs, plain old web pages. It also goes for every product and service that can be procured remotely: shoes, flowers, gadgets, hotel rooms, tutoring, investments. It even applies to people looking for a job or a date. How do you find each other? This is the defining problem of the Information Age, and machine learning is a big part of the solution.

They consolidated all voter information into a single database; combined it with what they could get from socialnetworking, marketing, and other sources; and set about predicting four things for each individual voter: how likely he or she was to support Obama, show up at the polls, respond to the campaign's reminders to do so, and change his or her mind about the election based on a conversation about a specific issue. Based on these voter models, every night the campaign ran 66,000 simulations of the election and used the results to direct its army of volunteers: whom to call, which doors to knock on, what to say.

For example, Google's business is based on guessing which web pages you're looking for when you type some keywords into the search box. Their key asset is massive logs of search queries people have entered in the past and the links they clicked on in the corresponding re
sults pages. But what do you do if someone types in a combination of keywords that's not in the log? And even if it is, how can you be confident that the current user wants the same pages as the previous ones?
How about we just assume
What we need is to learn concepts that are defined by a set of rules, not just a single rule, such as:
If you liked Star Wars, episodes IV–VI, you'll like Avatar.
If you liked Star Trek: The Next Generation and Titanic, you'll like Avatar.
If you're a member of the Sierra Club and read science-fiction books, you'll like Avatar.
Or:

After we learn each rule, we discard the positive examples that it accounts for, so the next rule tries to account for as many of the remaining positive examples as possible, and so on until all are accounted for. It's an example of "divide and conquer," the oldest strategy in the scientist's playbook. We can also improve the algorithm for finding a single rule by keeping some number n of hypotheses around, not just one, and at each step extending all of them in all possible ways and keeping the n best results.
Discovering rules in this way was the brainchild of Ryszard

"**Entities should not be multiplied beyond necessity," as the razor is often paraphrased, just means choosing the simplest theory that fits the data.**

In machine learning, **the technical terms for these are bias and variance. A clock that's always an hour late has high bias but low variance. If instead the clock alternates** erratically between fast and slow but on average tells the right time, it has high variance but low

bias. Suppose you're down at the pub with some friends, drinking

A **decision tree is like playing a game of twenty questions with an instance. Starting at the root, each node asks about the value of one attribute, and depending on the answer**, we follow one or another branch. When we arrive at a leaf, we read off the predicted concept. Each path from the root to a leaf corresponds to a rule. If this reminds you of those annoying phone menus you have to get through when you call customer service, it's not an accident: a phone menu is a decision tree. The computer on the other end of the line is playing a game of twenty questions with you to figure out what you want, and each menu is a question.

P(cause | effect) = P(cause) × P(effect | cause) / P(effect).
Replace cause by A and effect by B and omit the multiplication sign for brevity, and you get the ten-foot formula in the cathedral.

In the universe of all patients, the probability of fever is 20/100; in the universe of flu-stricken patients, it's 11/14. The probability that a patient has the flu and a fever is the fraction of patients that have the flu times the fraction of those that have a fever: P(flu, fever) = P(flu) × P(fever | flu) = 14/100 × 11/14 = 11/100. But we could equally well have done this the other way around: P(flu, fever) = P(fever) × P(flu | fever). Therefore, since they're both equal to P(flu, fever), P(fever) × P(flu | fever) = P(flu) × P(fever | flu). Divide both sides by P(fever), and you get P(flu | fever) = P(flu) × P(fever | flu) / P(fever). That's it! That's Bayes' theorem, with flu as the cause and fever as the effect.

Naïve Bayes is now very widely used. For example, it forms the basis of many spam filters. It all began when David Heckerman, a prominent Bayesian researcher who is also a medical doctor, had the idea of treating spam as a disease whose symptoms are the words in the e-mail: Viagra is a symptom, and so is free, but your best friend's first name probably signals a legit e-mail. We can then use **Naïve Bayes to classify e-mails into spam and nonspam, provided spammers generate e-mails by picking words at random.** That's a ridiculous assumption, of course: it would only be true if sentences had no syntax and no content. But that summer Mehran Sahami, then a Stanford graduate student, tried it out during an internship at Microsoft Research, and it worked great. When Bill Gates asked Heckerman how this could be, he pointed out that to identify spam you don't need to understand the details of the message: it's enough to get the gist of it by seeing which words it contains.

Joseph Schumpeter said that the economy evolves by cracks and leaps: S curves are the shape of creative destruction. The effect of financial gains and losses on your happiness follows an S curve, so don't sweat the big stuff. The probability that a random logical formula is satisfiable—the quintessential NP-complete problem— undergoes a phase transition from almost 1 to almost 0 as the formula's length increases. Statistical physicists spend their lives studying phase transitions.
In Hemingway's The Sun Also Rises, when Mike Campbell is asked how he went bankrupt, he replies, "Two ways. Gradually and then suddenly." The same could be said of Lehman Brothers. That's the essence of an S curve.

PageRank, the algorithm that gave rise to Google, is itself a Markov chain. Larry Page's idea was that web pages with many incoming links are probably more important than pages with few, and links from important pages should themselves count for more. This sets up an infinite

regress, but we can handle it with a Markov chain. Imagine a web surfer going from page to page by randomly following links: the states of this Markov chain are web pages instead of characters, making it a vastly larger problem, but the math is the same. A page's score is then the fraction of the time the surfer spends on it, or equivalently, his probability of landing on the page after wandering around for a long time.

Siri aside, you use an HMM every time you talk on your cell phone. That's because your words get sent over the air as a stream of bits, and the bits get corrupted in transit. The HMM then figures out the intended bits (hidden state) from the ones received (observations), which it should be able to do as long as not too many bits got mangled.

If the states and observations are continuous variables instead of discrete ones, the HMM becomes what's known as a Kalman filter. Economists use Kalman filters to remove noise from time series of quantities like GDP, inflation, and unemployment. The "true" GDP values are the hidden states; at each time step, the true value should be similar to the observed one, but also to the previous true value, since the economy seldom makes abrupt jumps. The Kalman filter trades off these two, yielding a smoother curve that still accords with the observations. When a missile cruises to its target, it's a Kalman filter that keeps it on track. Without it, there would have been no man on the moon.

In retrospect, we can see that Naïve Bayes, Markov chains, and HMMs are all special cases of Bayesian networks. The structure of Naïve Bayes is:

We can think of a Bayesian network as a "generative model," a recipe for probabilistically generating a state of the world: first decide independently whether there's a burglary and/or an earthquake, then based on that decide whether the alarm goes off, and then based on that whether Bob and Claire call. A Bayesian network tells a story: A happened, and it led to B; at the same time, C also happened, and B and C together caused D. To compute the probability of a particular story, we just multiply the probabilities of all of its different strands.

Learning the Bayesian way
Now that we know how to (more or less) solve the inference problem, we're ready to learn Bayesian networks from data, because for Bayesians learning is just another kind of probabilistic inference. All you have to do is apply Bayes' theorem with the hypotheses as the possible causes and the data as the observed effect:
P(**hypothesis | data) = P(hypothesis) × P(data | hypothesis) / P(da**

You don't need explicit ratings to do collaborative filtering, by the way. If Ken ordered a movie on Netflix, that means he expects to like it. So the "ratings" can just be ordered/not ordered, and two users are similar if they've ordered a lot of the same movies. Even just clicking on something implicitly shows interest in it. Nearest-neighbor works with all of the above. These days all kinds of algorithms are used to recommend items to users, but weighted k-nearest-neighbor was the first widely used one, and it's still hard to beat.
Recommender systems, as they're also called, are big business: a third of Amazon's business comes from its recommendations, as does three-quarters of Netflix's. It's a far cry from the early days of nearest-neighbor, when it was considered impractical because

In fact, no learner is immune to the curse of dimensionality. It's the second worst

problem in machine learning, after overfitting. The term curse of dimensionality was coined by Richard Bellman, a control theorist, in the fifties. He observed that control algorithms that worked fine in three dimensions became hopelessly inefficient in higher-dimensional spaces, such as when you want to control every joint in a robot arm or every knob in a chemical plant. But in machine learning the problem is more than just computational cost—it's that learning itself becomes harder and harder as the dimensionality goes up.
All is not lost, however. The first thing we can do is get rid of the irrelevant dimensions. Decision trees do this automatically by computing the information gain of each attribute and using only the most informative ones. For nearest-neighbor, we can accomplish something similar by first discarding all attributes whose information gain is below some threshold and then measuring similarity only in the reduced space. This is quick and good enough for some applications, but unfortunately it precludes learning many concepts, like exclusive-OR: if an attribute only says something about the class when combined with others, but not on its own, it will be discarded.

Superficially, an **SVM looks a lot like weighted k-nearest-neighbor: the frontier between the positive and negative classes is defined by a set of examples and their** weights, together with a similarity measure. A test example belongs to the positive class if, on average, it looks more like the positive examples than the negative ones. The average is weighted, and the SVM remembers only the key examples required to pin down the frontier.

Constrained optimization is the problem of maximizing or minimizing a function subject to constraints. The universe maximizes entropy subject to keeping energy constant. Problems of this type are widespread in business and technology. For example, we may want to maximize the number of widgets a factory produces, subject to the number of machine tools available, the widgets' specs, and so on. With SVMs, constrained optimization became crucial for machine learning as well. Unconstrained optimization is getting to the top of the mountain, and that's what gradient descent (or, in this case, ascent) does. Constrained optimization is going as high as you can while staying on the road. If the road goes up to the very top, the constrained and unconstrained problems have the same solution. More

The **most important question in any analogical learner is how to measure similarity. It could be as simple as Euclidean distance between** data points, or as complex as a whole program with multiple levels of subroutines whose final output is a similarity value. Either way, the similarity function controls how the learner generalizes from known examples to new ones. It's where we insert our knowledge of the problem domain into the

A cluster is a set of similar entities, or at a minimum, a set of entities that are more similar to each other than to members of other clusters. It's human nature to cluster things, and it's often the first step on the road to knowledge. When we look up at the night sky, we can't help seeing clusters

Principal-component analysis (PCA), as this process is known, is one of the key tools in the scientist's toolkit. You could say PCA is to unsupervised learning what linear regression is to the supervised variety. The famous hockey-stick curve of global warming, for example, is the result of finding the principal component of various temperature-related data series (tree rings, ice cores, etc.) and assuming it's the temperature. Biologists use PCA to summarize the expression levels of thousands of different genes into a few pathways. Psychologists have found that personality boils down to five dimensions—extroversion,

agreeableness, conscientiousness, neuroticism, and openness to experience—which

Instead of just recommending movies that users with similar tastes liked, it first projects both users and movies into a lower-dimensional "taste space" and recommends a movie if it's close to you in this space. That way it can find movies for you that you never knew you'd love.

After you've mastered your golf swing or tennis serve, you can spend years perfecting it, but all those years make less difference than the first few weeks did. You get better with practice, but not at a constant rate: at first you improve quickly, then not so quickly, then very slowly. Whether it's playing games or the guitar, the curve of performance improvement over time—how well you do something or how long it takes you to do it—has a very specific form:

This type of curve is called a power law, because performance varies as time raised to some negative power. For example, in the figure above, time to completion is proportional to the number of trials raised to minus two (or equivalently, one over the number of trials squared). Pretty much every human skill follows a power law, with different powers for different skills. (In contrast, Windows never gets faster with practice—something for Microsoft to work on.)

(seven plus or minus two, according to the classic paper by George Miller). Crucially, grouping things into chunks allows us to process much more information than we otherwise could. That's why telephone numbers have hyphens: 1-723-4583897 is *much easier to remember than 17234583897. Herbert Simon, Newell's longtime collaborator and AI cofounder, had earlier found that the main difference between novice and expert chess players is that novices perceive chess positions one piece at a time while experts see larger patterns involving multiple pieces. Getting better at chess mainly involves acquiring more and larger such chunks*. Newell and Rosenbloom hypothesized that a similar process is at work in all skill acquisition, not just chess.

If the background color of your e-commerce site's home page is currently blue and you're wondering whether making it red would increase sales, try it out on a hundred thousand randomly chosen customers and compare the results with those of the regular site. This technique, called A/B testing, was at first used mainly in drug trials but has since spread to many fields where data can be gathered on demand, from marketing to foreign aid. It can also be generalized to try many combinations of changes at once, without losing track of which changes lead to which gains (or losses). Companies like Amazon and Google swear by it; you've probably participated in thousands of A/B tests

Whether you read this book out of curiosity or professional interest, I hope you will share what you've learned with your friends and colleagues. Machine learning touches the lives of every one of us, and it's up to all of us to decide what we want to do with it. Armed with your new understanding of machine learning, you're in a much better position to think about issues like privacy and data sharing, the future of work, robot warfare, and the promise and peril of AI; and the more of us have this understanding, the more likely we'll avoid the pitfalls and find the right paths. That's the other big reason I wrote this book. The statistician knows that prediction is hard, especially about the future, and the computer scientist knows that the best way to predict the future is to invent it, butst and most, I thank my family for their love and support.

If you'd like to learn more about machine learning in general, one good place to start is online courses**. Of these, the closest in content to this book is, not coincidentally, the one I teach**

(www.coursera.org/course/machlearning). Two other options are Andrew Ng's course (www.coursera.org/course/ml) and Yaser Abu-Mostafa's (http://work.caltech.edu/telecourse.html). The next step is to read a textbook. The closest to this book, and one of the most accessible, is Tom Mitchell's Machine Learning* (McGraw-Hill, 1997). More up-to-date, but also more mathematical, are Kevin Murphy's Machine Learning: A Probabilistic Perspective* (MIT Press, 2012), Chris Bishop's Pattern Recognition and Machine Learning* (Springer, 2006), and An Introduction to Statistical Learning with Applications in R,* by Gareth James, Daniela Witten, Trevor Hastie, and Rob Tibshirani (Springer, 2013). My article "A few useful things to know about machine learning" (Communications of the ACM, 2012) summarizes some of the "folk knowledge" of machine learning that textbooks often leave implicit and was one of the starting points for this book.

can start from a number of open-source packages, such as Weka (www.cs.waikato.ac.nz/ml/weka). The two main machine-learning journals are Machine Learning and the Journal of Machine Learning Research. Leading machine-learning conferences, with yearly proceedings, include the International Conference on Machine Learning, the Conference on Neural Information Processing Systems, and the International Conference on Knowledge Discovery and Data Mining. A large number of machine-learning talks are a vailable on http://videolectures.net. The www. KDnuggets.com website is a one-stop shop for machine-learning resources, and you can sign up for its newsletter to keep up-to-date with the latest developments.

The textbook I learned AI from is Artificial Intelligence,* by Elaine Rich (McGraw-Hill, 1983). A current one is Artificial Intelligence: A Modern Approach, by Stuart Russell and Peter Norvig (3rd ed., Prentice Hall, 2010). Nils Nilsson's The Quest for Artificial Intelligence (Cambridge University Press, 2010) tells the story of AI from its earliest days.

Algorithms,* by Sanjoy Dasgupta, Christos Papadimitriou, and Umesh Vazirani (McGraw-Hill, 2008), is a concise introductory textbook on the subject. The