Chapter 0

Introduction[1]

Machine learning attempts to tell how to automatically find a good predictor based on past experiences. Although you might argue that machine learning has been around as long as statistics has, it really only became a separate topic in the 1990's. It draws its inspiration from a variety of academic disciplines, including computer science, statistics, biology, and psychology.

In this class we're going to look at some of the significant results from machine learning. One goal is to learn some of the techniques of machine learning, but also, just as significant, we are going to get a glimpse of the research front and the sort of approaches researchers have taken toward this very nebulous goal of automatically finding predictors.

Researchers have approached the general goal of machine learning from a variety of approaches. Before we delve into details about these, let's do a general overview of machine learning research. This also constitutes something of an outline of this text: We'll spend a chapter on each of the following topics.

0.1 Data mining

With the arrival of computing in all facets of day-to-day business, the amount of accessible data has exploded. Employers keep information about employees, businesses keep information about customers, hospitals keep information about patients, factories get information about instrument performance, scientists collect information about the natural world — and it's all stored in computers, ready to access in mass.

Data mining is gradually proving itself as an important tool for people who wish to analyze all this data for patterns. One of the most famous examples is from the late 1970s, when data mining proved itself as potentially important for both scientific and commercial purposes on a particular test application of diagnosing diseases in soybean plants [MC80].

First the researchers found an expert, who they interviewed for a list of rules about how to diagnose diseases in a soybean plant. Then they collected about 680 diseased soybean plants and determined about 35 pieces of data on each case (such as the month the plant was found to have a disease, the amount of recent precipitation, the size of the plant's seeds, the condition of its roots). They handed the plants to the expert to diagnose, and then they performed some data mining techniques to look for rules predicting the disease based on the characteristics they measured.

What they found is that the rules the expert gave during the interview were accurate only 70% of the time, while the rules discovered through data mining were accurate 97.5% of the time. Moreover, after revealing these rules to the expert, the expert reportedly was impressed enough to adopt some of the rules in place of the ones given during the interview!

[1]*A Survey of Machine Learning*, by Carl Burch. ©2001, Carl Burch. All rights reserved.

In this text, we'll see a few of the more important machine learning techniques used in data mining, as well as surrounding issues that apply regardless of the learning algorithm. We'll emerge from this familiar with much of the established machine learning results and prepared to study less polished research.

0.2 Neural networks

Cognitive science aims to understand how the human brain works. One important part of cognitive science involves simulating models of the human brain on a computer to learn more about the model and also potentially to instill additional intelligence into the computer.

Though scientists are far from understanding the brain, machine learning has already reaped the reward of *artificial neural networks* (ANNs). ANNs are deployed in a variety of situations. One of the more impressive is ALVINN, a system that uses an ANN to steer a car on a highway [Pom93]. ALVINN consists of an array of cameras on the car, whose inputs are fed into a small ANN whose outputs control the steering wheel. The system has been tested going up to 70 miles per hour over 90 miles of a public divided highway (with other vehicles on the road).

Here we'll look briefly at human neurons and how artificial neurons model their behavior. Then we'll see how they might be networked together to form an artificial neural network that improves with training.

0.3 Reinforcement learning

Through this point, we'll have worked exclusively with systems that need immediate feedback from their actions in order to learn. This is **supervised learning**, which though useful is a limited goal.

Reinforcement learning (sometimes called **unsupervised learning**) refers to a brand of learning situation where a machine should learn to behave in situations where feedback is not immediate. This is especially applicable in robotics situations, where you might hope for a robot to learn how to accomplish a certain task in the same way a dog learns a trick, without explicit programming.

Probably the most famous success story from reinforcement learning is TD-Gammon, a program that learns to play the game of Backgammon [Tes95]. TD-Gammon uses a neural network coupled with reinforcement learning techniques. After playing against itself for 1.5 million games, the program learned enough to rank among the world's best backgammon players (including both humans and computers).

Reinforcement learning is much more challenging than supervised learning, and researchers still don't have a good grasp on it. We'll see a few of the proposed techniques, though, and how they can be applied in situations like the one that TD-Gammon tackles.

0.4 Artificial life

Finally, **artificial life** seeks to emulate living systems with a computer. Our study of artificial life will concentrate on genetic algorithms, where systems loosely based on evolution are simulated to see what might evolve. They hope to evolve very simple behavior, like that of amoebas, thus gaining a greater understanding of how evolution works and what effects it has.

In a sense, the evolutionary process *learn* — though usually the verb we use is *adapt*. If you understanding learning as improving performance based on past experience, the word *learn* has a similar denotation to *adapt*, even if the connotation is different.

Genetic algorithms appear to be a promising technique for learning from past experience, even outside simulations of pseudo-biological. We'll look at this technique, and then we'll look at its use in attempting to evolve simple behaviors.

Chapter 1

Data mining

Data is abundant in today's society. Every transaction, every accomplishment gets stored away somewhere. We get much more data than we could ever hope to analyze by hand. A natural hope is to analyze the data by computer. **Data mining** refers to the various tasks of analyzing stored data for patterns — seeking clusters, trends, predictors, and patterns in a mass of stored data.

For example, many grocery stores now have customer cards, rewarding frequent users of the grocery store with discounts on particular items. The stores gives these cards to encourage customer loyalty and to collect data — for, with this cards, they can track a customer between visits to the store and potentially mine the collected data to determine patterns of customer behavior. This is useful for determining what to promote through advertisement, shelf placement, or discounts.

A large banking corporation makes many decisions about whether to accept a loan application or not. On hand is a variety of information about the applicant — age, employment history, salary, credit history — and about the loan application — amount, purpose, interest rate. Additionally, the bank has the same information about thousands of past loans, plus whether the loan proved to be a good investment or not. From this, the bank wants to know whether it should make the loan. Data mining can potentially improve the loan acceptance rate without sacrificing on the default rate, profiting both the bank and its customers.

1.1 Predicting from examples

We'll emphasize a particular type of data mining called **predicting from examples**. In this scenario, the algorithm has access to several **training examples**, representing the past history. Each training example is a vector of values for the different **attributes** measured. In our banking application, each example represents a single loan application, represented by a vector holding the customer age, customer salary, loan amount, and other characteristics of the loan application. Each example has a **label**, which in the banking application might be a rating of how well the loan turned out.

The learning algorithm sees all these labeled examples, and then it should produce a **hypothesis** — a new algorithm that, given a new vector as an input, produces a prediction of its label. (That a learning algorithm produces another algorithm may strike you as odd. But, given the variety of types of hypotheses that learning algorithms produce, this is the best we can do formally.)

In Section 1.1, we briefly looked at some data mining results for soybean disease diagnosis by Michalski and Chilausky [MC80]. Figure 1.1 illustrates a selection of the data they used. (I've taken just seven of the 680 plants and just four of the 35 attributes.) There are six training examples here, each with four attributes (plant growth, stem condition, leaf-spot halo, and seed mold) and a label (the disease). We would feed those into the learning algorithm, and it would produce a hypothesis that labels any plant with a supposed disease.

	plant growth	stem condition	halo on leaf spots	mold on seed	disease
Plant A	normal	abnormal	no	no	frog eye leaf spot
Plant B	abnormal	abnormal	no	no	herbicide injury
Plant C	normal	normal	yes	yes	downy mildew
Plant D	abnormal	normal	yes	yes	bacterial pustule
Plant E	normal	normal	yes	no	bacterial blight
Plant F	normal	abnormal	no	no	frog eye leaf spot
Plant Q	normal	normal	no	yes	???

Figure 1.1: A prediction-from-examples problem of diagnosing soybean diseases.

	sepal length	sepal width	petal length	petal width	species
Iris A	4.7	3.2	1.3	0.2	*setosa*
Iris B	6.1	2.8	4.7	1.2	*versicolor*
Iris C	5.6	3.0	4.1	1.3	*versicolor*
Iris D	5.8	2.7	5.1	1.9	*virginica*
Iris E	6.5	3.2	5.1	2.0	*virginica*
Iris Q	5.8	2.7	3.9	1.2	???

Figure 1.2: A prediction-from-examples problem of Iris classification.

Plant Q, representing a new plant whose disease we want to diagnose, illustrates what we might feed to the hypothesis to arrive at a prediction.

In the soybean data, each of the four attributes has only two possible values (either normal and abnormal, or yes and no); the full data set has some attributes with more possible values (the month found could be any month between April or October; the precipitation could be below normal, normal, or above normal). But each attribute has just a small number of possible values. Such attributes are called **discrete attributes**.

In some domains, attributes may be **numeric** instead. A numeric attribute has several possible values, in a meaningful linear order. Consider the classic data set created by Fisher, a statistician working in the mid-1930s [Fis36]. Fisher measured the sepals and petals of 150 different irises, and labeled them with the specific species of iris. For illustration purposes, we'll just work with the five labeled examples of Figure 1.2 and the single unlabeled example whose identity we want to predict.

These two examples of data sets come from actual data sets, but they are simpler than what one normally encounters in practice. Normally, data has many more attributes and several times the number of examples. Moreover, some data will be missing, and attributes are usually mixed in character — some of them will be discrete (like a loan applicant's state of residence) while others are numeric (like the loan applicant's salary). But these things complicate the basic ideas, which is what we want to look at here, so we'll keep with the idealized examples.

1.2 Techniques

We're going to look at three important techniques for data mining. The first two — linear regression and nearest-neighbor search — are best suited for numeric data. The last, ID3, is aimed at discrete data.

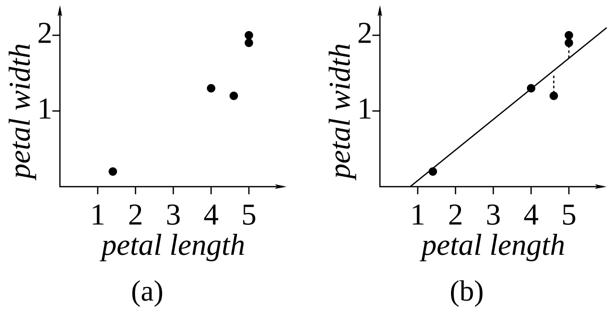


Figure 1.3: Fitting a line to a set of data.

1.2.1 Linear regression

Linear regression is one of the oldest forms of machine learning. It is a long-established statistical technique that involves simply fitting a line to some data.

Single-attribute examples The easiest case for linear regression is when the examples have a single numeric attribute and a numeric label; we'll look at this case first. Say we have n examples, where the attribute for each example is called x_i, and the label for each is y_i. We can envision each example as being a point in 2-dimensional space, with an x-coordinate of x_i and a y-coordinate of y_i. (See Figure 1.3(a).)

Linear regression would seek the line $f(x) = mx + b$ (specifying the prediction $f(x)$ for any given single-attribute example $\langle x \rangle$) that minimizes the **sum-of-squares error** for the training examples,

$$\sum_{i=1}^{n} (y_i - f(x_i))^2 .$$

(See Figure 1.3(b).) The quantity $|y_i - f(x_i)|$ is the distance from the value predicted by the hypothesis line to the actual value — the error of the hypothesis for this training example. Squaring this value gives greater emphasis to larger errors and saves us dealing with complicated absolute values in the mathematics.

For example, we might want to predict the petal width of an iris given its petal length using the data of Figure 1.2. Here x_1 is 1.3 (the petal length of Iris A) and y_1 is 0.2. From Iris B, we get $x_2 = 4.7$ and $y_2 = 1.2$. We get similar data from Iris C, Iris D, and Iris E. Figure 1.3 graphs the points.

With a little bit of calculus, it's not too hard to compute the exact values of m and b that minimize the sum-of-squares error. We'll skip the derivation and just show the result. Let $\bar{x}$ be the average x value ($(\sum_i x_i)/n$) and $\bar{y}$ be the average y value ($(\sum_i y_i)/n$). The optimal choices for m and b are

$$m = \frac{(\sum_i x_i y_i) - n\bar{x}\bar{y}}{(\sum_i x_i^2) - n\bar{x}^2}, b = \bar{y} - m\bar{x} .$$

For the irises, we compute that $\sum_i x_i = 20.3$ and hence $\bar{x} = 4.05$, $\sum_i y_i = 6.6$ and hence $\bar{y} = 1.32$, $\sum_i x_i y_i = 31.12$, $\sum_i x_i^2 = 92.61$. Knowing these, we compute m to be $(31.12 - 5 \cdot 4.05 \cdot 1.32)/(92.61 - 5 \cdot 4.05^2) = 4.39/10.5975 \approx 0.4142$ and b to be $1.32 - 0.4142 \cdot 4.05 = -0.3577$. So our hypothesis is that, if the petal length is x, then the petal width will be

$$0.4142x - .3577 .$$

On Iris Q, with a petal length of 3.9, this hypothesis would predict a petal width of 1.26, not too far from the actual value of 1.2. (On the other hand, the hypothesis would predict a negative *petal width for a very short petal, which is clearly not appropriate.)*

Multiple-attribute examples When each piece of training data is a vector of *several* attributes, the problem becomes more complicated. What we're going to do is to expand the single-attribute example by expanding the dimensions of the space. If each example has just two attributes, we could view each labeled example in three-dimensional space, with an x-coordinate corresponding to the first attribute, the y-coordinate corresponding to the second attribute, and the z-coordinate corresponding to the label. We'd look for a plane $f(x, y) = m_x x + m_y y$ that minimizes the sum-of-squares error.

For a general number of attributes d, we'll view each labeled example as a point in $(d+1)$-dimensional space, with a coordinate for each attribute, plus a coordinate for the label. We'll look for a d-dimensional hyperplane $f(x_1, x_2, \ldots, x_d) = \sum_{i=1}^{d} m_i x_i$ that minimizes the sum-of-squares error.

This is slightly different than the two-dimensional case because this $f(x)$ is forced to go through the origin $(0, 0, \ldots, 0)$. Forcing this keeps the mathematics prettier. If we want to be able to learn a hyperplane that doesn't necessarily go through the origin, we can get around this limitation by simply inserting an additional attribute to every vector that is always 1 (so that $m_i x_i$ for that coordinate is just the constant m_i).

For the iris example, we're in 6-dimensional space. We're adding the extra always-1 attribute, giving us a total of $d = 5$ attributes in each example, and then we have the label. The label needs to be numeric, though the iris example labels aren't. What we'll do is say that the label is 1 if the species is versicolor *and 0 otherwise. Thus if the predicted value is large (at least 0.5), the hypothesis is that the iris is probably* versicolor *and if the predicted value is small (less than 0.5), the hypothesis is that it is probably not. Thus Iris A is at the point $(1, 4.7, 3.2, 1.3, 0.2, 0)$. Iris B is at the point $(1, 6.1, 2.8, 4.7, 1.2, 1)$.*

We'll use the notation x_{ij} to refer to the ith example's jth attribute value and the notation y_i to refer to the ith example's label. Thus our examples are as follows.

$$\begin{array}{ccccccccc}
(& x_{11}, & x_{12}, & x_{13}, & \ldots, & x_{1d}, & y_1 &) \\
(& x_{21}, & x_{22}, & x_{23}, & \ldots, & x_{2d}, & y_2 &) \\
 & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \\
(& x_{n1}, & x_{n2}, & x_{n3}, & \ldots, & x_{nd}, & y_n &)
\end{array}$$

We want to find a set of coefficients m_i so that the function $f(x) = \sum_{i=1}^{d} m_i x_i$ as closely approximates the y_i as possible (still using the sum-of-squares error). To compute this $f(x)$, it turns out that we need to solve a set of equations.

$$\begin{array}{ccccccccc}
\left(\sum_i x_{i1} x_{i1}\right) m_1 & + & \left(\sum_i x_{i1} x_{i2}\right) m_2 & + & \cdots & + & \left(\sum_i x_{i1} x_{id}\right) m_d & = & \sum_i x_{i1} y_i \\
\left(\sum_i x_{i2} x_{i1}\right) m_1 & + & \left(\sum_i x_{i2} x_{i2}\right) m_2 & + & \cdots & + & \left(\sum_i x_{i2} x_{id}\right) m_d & = & \sum_i x_{i2} y_i \\
\vdots & & \vdots & & \ddots & & \vdots & & \vdots \\
\left(\sum_i x_{id} x_{i1}\right) m_1 & + & \left(\sum_i x_{id} x_{i2}\right) m_2 & + & \cdots & + & \left(\sum_i x_{id} x_{id}\right) m_d & = & \sum_i x_{id} y_i
\end{array}$$

Here we have d equations and d unknowns (namely, m_1 through m_d). After we solve for the m_i, the best-fit hyperplane is the function $f(x) = \sum_i m_i x_i$. Given an example $\langle x_1, \ldots, x_d \rangle$ for which we want to make a prediction, we would predict $f(\langle x_1, \ldots, x_d \rangle)$.

So to compute the best-fit hyperplane for our versicolor *labeling, we'll have to find our set of equa-*

tions. To do that, *we need to compute a lot of sums. Here we go.*

$$\begin{array}{lclcr}
\sum_i x_{i1}x_{i1} & = & 1^2+1^2+1^2+1^2+1^2 & = & 5.0 \\
\sum_i x_{i1}x_{i2} & = & 1\cdot 4.7+1\cdot 6.1+1\cdot 5.6+1\cdot 5.8+1\cdot 6.5 & = & 28.7 \\
\sum_i x_{i1}x_{i3} & = & 1\cdot 3.2+1\cdot 2.8+1\cdot 3.0+1\cdot 2.7+1\cdot 3.2 & = & 14.9 \\
\sum_i x_{i1}x_{i4} & = & 1\cdot 1.3+1\cdot 4.7+1\cdot 4.1+1\cdot 5.1+1\cdot 5.1 & = & 20.3 \\
\sum_i x_{i1}x_{i5} & = & 1\cdot 0.2+1\cdot 1.2+1\cdot 1.3+1\cdot 1.9+1\cdot 2.0 & = & 6.6 \\
\sum_i x_{i1}y_i & = & 1\cdot 0+1\cdot 1+1\cdot 1+1\cdot 0+1\cdot 0 & = & 2.0 \\
\sum_i x_{i2}x_{i2} & = & 4.7^2+6.1^2+5.6^2+5.8^2+6.5^2 & = & 166.55 \\
\sum_i x_{i2}x_{i3} & = & 4.7\cdot 3.2+6.1\cdot 2.8+5.6\cdot 3.0+5.8\cdot 2.7+6.5\cdot 3.2 & = & 85.38 \\
\sum_i x_{i2}x_{i4} & = & 4.7\cdot 1.3+6.1\cdot 4.7+5.6\cdot 4.1+5.8\cdot 5.1+6.5\cdot 5.1 & = & 120.47 \\
\sum_i x_{i2}x_{i5} & = & 4.7\cdot 0.2+6.1\cdot 1.2+5.6\cdot 1.3+5.8\cdot 1.9+6.5\cdot 2.0 & = & 39.56 \\
\sum_i x_{i2}y_i & = & 4.7\cdot 0+6.1\cdot 1+5.6\cdot 1+5.8\cdot 0+6.5\cdot 0 & = & 11.7 \\
\sum_i x_{i3}x_{i3} & = & 3.2^2+2.8^2+3.0^2+2.7^2+3.2^2 & = & 44.61 \\
\sum_i x_{i3}x_{i4} & = & 3.2\cdot 1.3+2.8\cdot 4.7+3.0\cdot 4.1+2.7\cdot 5.1+3.2\cdot 5.1 & = & 59.71 \\
\sum_i x_{i3}x_{i5} & = & 3.2\cdot 0.2+2.8\cdot 1.2+3.0\cdot 1.3+2.7\cdot 1.9+3.2\cdot 2.0 & = & 19.43 \\
\sum_i x_{i3}y_i & = & 3.2\cdot 0+2.8\cdot 1+3.0\cdot 1+2.7\cdot 0+3.2\cdot 0 & = & 5.8 \\
\sum_i x_{i4}x_{i4} & = & 1.3^2+4.7^2+4.1^2+5.1^2+5.1^2 & = & 92.61 \\
\sum_i x_{i4}x_{i5} & = & 1.3\cdot 0.2+4.7\cdot 1.2+4.1\cdot 1.3+5.1\cdot 1.9+5.1\cdot 2.0 & = & 31.12 \\
\sum_i x_{i4}y_i & = & 1.3\cdot 0+4.7\cdot 1+4.1\cdot 1+5.1\cdot 0+5.1\cdot 0 & = & 8.8 \\
\sum_i x_{i5}x_{i5} & = & 0.2^2+1.2^2+1.3^2+1.9^2+2.0^2 & = & 10.78 \\
\sum_i x_{i5}y_i & = & 0.2\cdot 0+1.2\cdot 1+1.3\cdot 1+1.9\cdot 0+2.0\cdot 0 & = & 2.5 \\
\sum_i x_{i6}y_i & = & 0^2+1^2+1^2+0^2+0^2 & = & 2.0
\end{array}$$

From these we derive a set of five equations with five unknowns.

$$\begin{array}{rcrcrcrcrcr}
5.0m_1 & + & 28.70m_2 & + & 14.90m_3 & + & 20.30m_4 & + & 5.50m_5 & = & 2.0 \\
28.7m_1 & + & 166.55m_2 & + & 85.38m_3 & + & 120.47m_4 & + & 39.56m_5 & = & 11.7 \\
14.9m_1 & + & 85.38m_2 & + & 44.61m_3 & + & 59.71m_4 & + & 19.43m_5 & = & 5.8 \\
20.3m_1 & + & 120.47m_2 & + & 59.71m_3 & + & 92.61m_4 & + & 31.12m_5 & = & 8.8 \\
5.5m_1 & + & 39.56m_2 & + & 19.43m_3 & + & 31.12m_4 & + & 10.78m_5 & = & 2.5
\end{array}$$

*Now we solve this to get the hyperplane hypothesis. (*You *can try to do it by hand, but at this point I broke down and went to a computer.) The answer turns out to be the following.*

$$f(x_1,x_2,x_3,x_4,x_5) = 0.308x_1 + 0.736x_2 - 1.00x_3 - 0.278x_4 - 0.020x_5$$

For Iris Q, the predicted answer is

$$0.308\cdot 1 + 0.736\cdot 5.8 - 1.00\cdot 2.7 - 0.278\cdot 3.9 - 0.020\cdot 1.2 = 0.767\ .$$

Thus linear regression indicates that we are fairly confident that Iris Q is versicolor *(as indeed it is).*

Analysis Linear regression is really best suited for problems where the attributes and labels are all numeric and there is reason to expect that a linear function will approximate the problem. This is rarely a reasonable expectation — linear functions are just too restricted to represent a wide variety of hypotheses.
Advantages:

- Has real mathematical rigor.

- Handles irrelevant attributes somewhat well. (Irrelevant attributes tend to get a coefficient close to zero in the hyperplane's equation.)
- Hypothesis function is easy to understand.

Disadvantages:

- Oversimplifies the classification rule. (Why should we expect a hyperplane to be a good approximation?)
- Difficult to compute.
- Limited to numeric attributes.
- Doesn't at all represent how humans learn.

1.2.2 Nearest neighbor

Our second approach originates in the idea that given a query, the training example that is most similar to it probably has the same label. To determine what we mean by most "similar", we have to design some sort of **distance function**. In many cases, the best choice would be the **Euclidean distance function**, the straight-line distance between the two points $\langle x_{i1}, x_{i2}, \ldots, x_{id} \rangle$ and $\langle x_{j1}, x_{j2}, \ldots, x_{jd} \rangle$ in d-dimensional space. The formula for this would be

$$\sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{id} - x_{jd})^2}\,.$$

Another common possibility is to use the **Manhattan distance function**,

$$|x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \cdots + |x_{id} - x_{jd}|\ .$$

In practice, people generally go for the Euclidean distance function. They don't really have much mathematical reasoning to back this up; it just seems to work well in general.

In the Iris example, we'll compute the Euclidean distance from each of the vectors for the training examples to the query vector representing Iris Q.

example	*distance*		*label*
Iris A	$\sqrt{(4.7-5.8)^2+(3.2-2.7)^2+(1.3-3.9)^2+(0.2-1.2)^2}$	$= 3.04$	setosa
Iris B	$\sqrt{(6.1-5.8)^2+(2.8-2.7)^2+(4.7-3.9)^2+(1.2-1.2)^2}$	$= 0.86$	versicolor
Iris C	$\sqrt{(5.6-5.8)^2+(3.0-2.7)^2+(4.1-3.9)^2+(1.3-1.2)^2}$	$= 0.42$	versicolor
Iris D	$\sqrt{(5.8-5.8)^2+(2.7-2.7)^2+(5.1-3.9)^2+(1.9-1.2)^2}$	$= 1.39$	virginica
Iris E	$\sqrt{(6.5-5.8)^2+(3.2-2.7)^2+(5.1-3.9)^2+(2.0-1.2)^2}$	$= 1.68$	virginica

Since Iris C is the closest, the nearest-neighbor algorithm would predict that Iris Q has the same label as Iris C: versicolor.

There are two common refinements to the nearest-neighbor algorithm to address common issues with the data. The first is that the the raw Euclidean distance overemphasizes attributes that have broader ranges. In the Iris example, the petal length is given more emphasis than the petal width (since the length varies up to four centimeters, while the width only varies up to two centimeters). This is easy to fix by scaling each attribute by the maximum difference in the attribute values, to ensure that the distance between two attribute values in the same attribute never exceeds 1.

$$v_{ij} = \frac{x_{ij}}{\max_k x_{kj} - \min_k x_{kj}}$$

By applying the distance function to the v_{ij} instead of the x_{ij}, this artificial overemphasis disappears.

The second common refinement is to address the issue of **noise** — typically, a small but unknown fraction of the data might be mislabeled or unrepresentative. In the iris example, Iris C may not *really* be *versicolor*. Or perhaps Iris C's dimensions are uncharacteristic of *versicolor* specimens. We can get around this using the law of large numbers: Select some number k and use the k nearest neighbors. The prediction can be the average of these k nearest neighbors if the label values are numeric, or the plurality if the label values are discrete (breaking ties by choosing the closest).

If we choose k to be 3, we find that the three closest irises to Iris Q are Irises B, C, and D, of which two are versicolor *and one is* virginica. *Therefore we still predict that Iris Q is* versicolor.

Analysis The nearest-neighbor algorithm and its variants are particularly well-suited to **collaborative filtering**, where a system is to predict a given person's preference based on other people's preferences. For example, a movie Web site might ask you to rate some movies and then try to find movies you'd like to see. Here, each attribute is a single movie that you have seen, and the Web site looks for people whose movie preferences are close to yours and then predicts movies that these neighbors liked but that you have not seen. Or you might see this on book-shopping sites, where the site makes book recommendations based on your past order history.

Collaborative filtering fits into the nearest-neighbor search well because attributes tend to be numeric and similar in nature, so it makes sense to give them equal weight in the distance computation.

Advantages:

- Represents complex spaces very well.
- Easy to compute.

Disadvantages:

- Does not handle many irrelevant attributes well. If we have lots of irrelevant attributes, the distance between examples is dominated by the differences in these irrelevant attributes and so becomes meaningless.
- Still doesn't look much like how humans learn.
- Hypothesis function is too complex to describe easily.

1.2.3 ID3

Linear regression and nearest-neighbor search don't work very well when the data is discrete — they're really designed for numeric data. **ID3** is designed with discrete data in mind.

ID3's goal is to generate a decision tree that seems to describe the data. Figure 1.4 illustrates one decision tree. Given a vector, the decision tree predicts a label. To get its prediction, we start at the top and work downward. Say we take Plant Q. We start at the top node. Since Plant Q's stem is normal, we go to the right. Now we look for mold on Plant Q's seeds, and we find that it has some, so we go to the left from there. Finally we examine the spots on Plant Q's leaves; since they don't have yellow halos, we go to the right and conclude that Plant Q must have downy mildew.

Decision trees are well-suited for discrete data. They represent a good compromise between simplicity and complexity. Recall that one of our primary complaints about linear regression was that its hypothesis was too constricted to represent very many types of data, while one of our primary complaints about nearest-neighbor search was that its hypothesis was too complex to be understandable. Decision trees are easy to interpret but can represent a wide variety of functions.

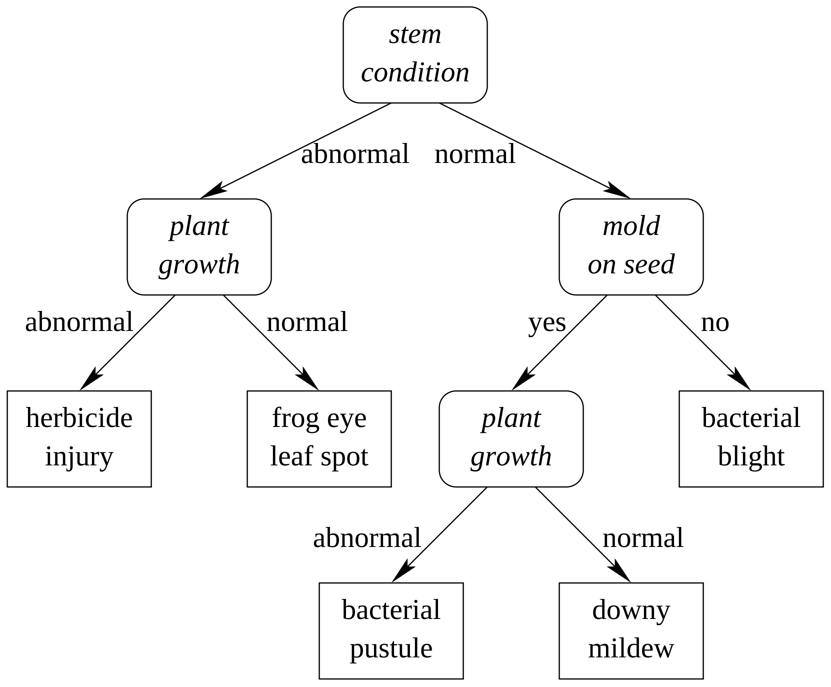
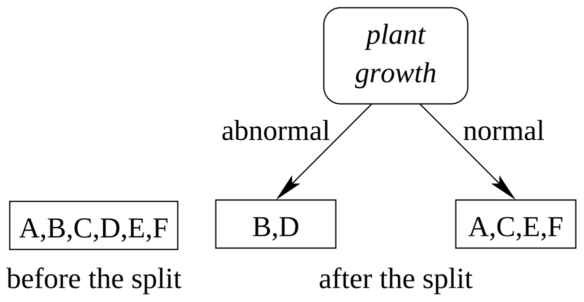


Figure 1.4: A decision tree example

Given a set of data, the goal of ID3 is to find a "good" decision tree that represents it. By *good*, the general goal is to find a small decision tree that approximates the true label pretty well.

Constructing the tree automatically ID3 follows a simple technique for constructing such a decision tree. We begin with a single node containing all the training data. Then we continue the following process: We find some node containing data with different labels, and we *split* it based on some attribute we select. By *split*, I mean that we take the node and replace it with new nodes for each possible value of the chosen attribute. For example, with the soybean data, if we have a node containing all the examples and choose to split on plant growth, the effect would be as follows.



We stop splitting nodes when every node of the tree is either labeled unanimously or contains indistinguishable vectors.

How does ID3 decide on which attribute to split a node? Before we answer this, we need to define the entropy of a set of examples. The **entropy** of a set S is defined by the following equation.

$$Entropy(S) = \sum_{\text{labels } \ell} -p_\ell \ln p_\ell$$

Here p_ℓ is the fraction of points in the set with the label ℓ.

In the training data of Figure 1.1, there are five labels: $2/6$ of the examples have frog eye leaf spot, and $1/6$ of the examples have each of the four other diseases. So the entropy is

$$-\frac{2}{6}\ln\frac{2}{6} + -\frac{1}{6}\ln\frac{1}{6} + -\frac{1}{6}\ln\frac{1}{6} + -\frac{1}{6}\ln\frac{1}{6} + -\frac{1}{6}\ln\frac{1}{6} = 1.5607\,.$$

This entropy is rather large, quantifying the fact that the set isn't labeled consistently at all.

The entropy is a weird quantity that people use more or less just because it works. When the entropy is small, this indicates that things are labeled pretty consistently. As an extreme example, if everything has the same label ℓ, then since $p_\ell = 1$ the entropy would be just $-p_\ell \ln p_\ell = 0$. We're aiming for a small decision tree where the total entropy of all the leaves is as small as possible.

Now let's look at what happens when we split a node. This splits the set of examples it represents into pieces based on the values of the attribute. To quantify how good the split is, we define the **gain** of splitting a node on a particular attribute: It is the entropy of the old set it represented, minus the weighted sum of the entropies of the new sets. Say S represents the old set, and S_v represents the examples of S with the value v for the attribute under consideration. The gain would be

$$\mathit{Entropy}(S) - \sum_{\text{values } v} \frac{|S_v|}{|S|}\mathit{Entropy}(S_v)\,.$$

At the beginning of the algorithm, all the examples are in a single node. Let's consider splitting on the plant growth. We just computed the entropy of all six training examples to be 1.5607. After we split on plant growth, we get two sets of plants: B and D have abnormal plant growth (this set has entropy $-(1/2)\ln(1/2) + -(1/2)\ln(1/2) = 0.6931$); and A, C, E, and F have normal plant growth (this set has entropy 1.0397). So the gain of splitting on plant growth is

$$1.5607 - \left(\frac{2}{6}0.6931 + \frac{4}{6}1.0397\right) = 0.6365\,.$$

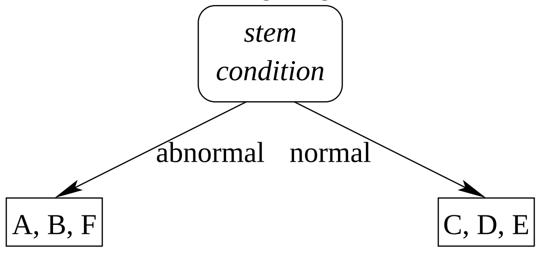
We can do similar calculations to compute the gain for stem condition instead. This divides the plants into a set of A, B, and F (entropy 0.6365) and a set of C, D, and E (entropy 1.0986). The gain of splitting on stem condition is

$$1.5607 - \left(\frac{3}{6}0.6365 + \frac{3}{6}1.0986\right) = 0.6931$$

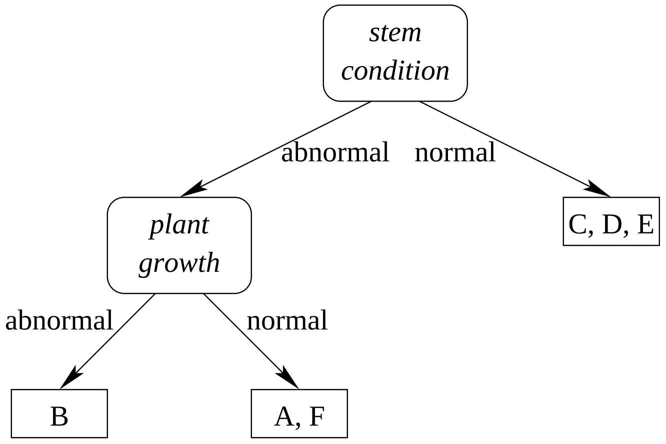
This is larger than the gain for plant growth, so splitting on stem condition is preferable.

After computing the gains of splitting for each of the attributes, we choose the attribute that gives the largest gain and then split on it, giving us a new tree. We continue splitting nodes until the examples in every node either have identical labels or indistinguishable attributes.

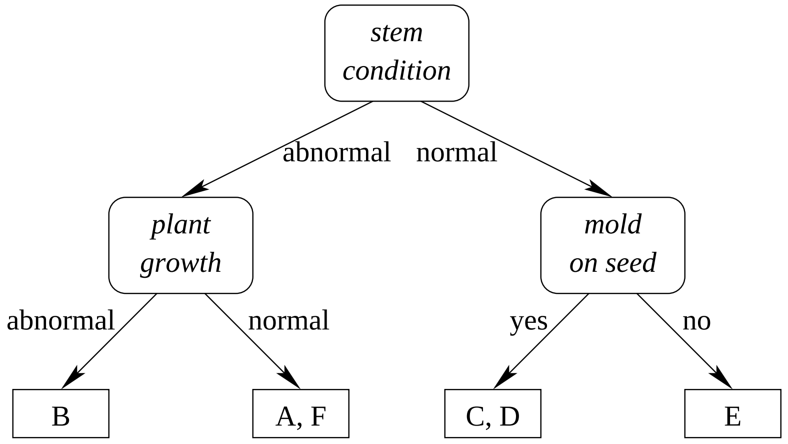
We would also consider splitting on leaf-spot halos (gain of 0.6931) and splitting on seed mold (gain of 0.6367). Of these, we could go with either stem condition or leaf-spot halos: They both have the same gain, 0.6931. We'll choose stem condition, giving us a new tree.



We repeat the process for each of the remaining sets. We first consider Plants A, B, and F. They don't have the same disease, so we'll look for an attribute on which to split them. That turns out to be easy, as they disagree only on the plant-growth attribute (none of them have leaf-spot halos or seed mold), so we'll split that node based on plant growth. (The gain of this would be 0.6365, while splitting on leaf-spot halos or seed mold gives a gain of 0.) Now we have the following tree.



The set of A and F isn't a problem, as they both have the same disease. So the next set we'll consider for splitting is C, D, and E. There, the gain for splitting based on plant growth is $1.0986 - ((2/3)0.6931 + (1/3)0) = 0.6365$. *The gain for splitting on leaf-spot halos is 0, since they all have leaf-spot halos. The gain for splitting on seed mold is* $1.0986 - ((2/3)0.6931 + (1/3)0) = 0.6365$. *We could go for either plant growth or seed mold; say we choose seed mold.*



Finally, we consider how to split Plants C and D. Splitting on plant growth gives a gain of 0.6931, while splitting on leaf-spot halos gives a gain of 0. We must split on plant growth, giving us the tree of Figure 1.4.

Analysis **Advantages:**

- Represents complex spaces well.
- Generates a simple, meaningful hypothesis.
- Filters out irrelevant data.

Disadvantages:

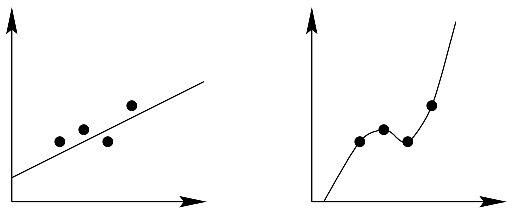


Figure 1.5: Overfitting some data.

- Doesn't handle numeric attributes well.
- Still doesn't look much like how humans learn.

1.3 General issues

There are a number of overarching data mining issues that apply regardless of the technique. The importance of these issues makes them worth studying separately. We'll look at just a few.

Data selection Choosing the data is an important element of applying data mining techniques. Of course we want the data to have integrity — though there's a natural tradeoff between integrity and quantity that we have to balance. We certainly need a good sample of data, or else the learned hypothesis won't be worth much. But we also need a large sample to work from.

Less obvious is the importance of a selecting good attributes. Sometimes we need to do some preprocessing to get good data. For example, with loan applications, the applicant's date of birth is probably what is really stored on the computer, but really this isn't significant to the application (neglecting astrological effects) — what's important is the applicant's age. So, though the database probably holds the birthday, we should compute the age to give to the learning algorithm. We can see a similar thing happening in the soybean data (Figure 1.1): Instead of giving the plant height (which after all, coupled with the month the plant was found implies whether growth is stunted), the data set just includes a feature saying whether plant growth is normal or abnormal. The researchers here did some preprocessing to simplify the task for the learner.

Overfitting In applying a machine learning technique, we need to be careful to avoid **overfitting**. This occurs when the algorithm adapts very carefully to the specific training data without improving general performance. For example, consider the two graphs in Figure 1.5. Although the graph at the right fits the data perfectly, it's likely that the graph at left is a better hypothesis.

Overfitting applies to just about any learning algorithm. ID3 is particularly prone to overfitting, as it continues growing the tree until it fits the data perfectly. Machine learning researchers have ways of working around this, but they get rather complicated, and so we're choosing to skip their approaches.

There's a tradeoff: Do we go for the perfect fit (which may be an overfit), or do we settle for a simpler hypothesis that seems to be pretty close? The answer is that this is part of the art of applying machine learning techniques to data mining. But an aid to this is to be able to evaluate the error of a given hypothesis.

Evaluating hypotheses Once we get a hypothesis from a learning algorithm, we often need to get some sort of estimate of how good it is. The most typical measure is the **error**: the probability that the hypothesis predicts the wrong label for a randomly chosen new example.

It's very tempting to feed all the data we have into the learning algorithm, but if we need to report the hypothesis error, this is a mistake. We also need to use some data to compute the error, and we can't reuse the training data for this computation. (This would be analogous to giving students all the answers to a test the day before: Of course they'll do well on the test, but what have they learned?)

So in situations where we need to compute the error, we separate the data into two sets — the **training set**, which holds the examples we give to the learning algorithm — and the **test set**, which we use for computing the error. The error we report would be the fraction of the examples in the test set on which the learning algorithm's hypothesis predicts wrongly. Typically two-thirds of the data might go into the training set and a third into the test set, to give a reasonable tradeoff on the accuracy of the hypothesis and the accuracy of the reported error.

In many situations, the data just isn't plentiful enough to allow this. U S Presidential elections would be a good example: It's not as if we can go out and generate new elections, so we're stuck with the handful we have. If we want to apply a learning algorithm to past polling data and their effect on the final result, we want to use all the data we can find. Machine learning has proposed several techniques for both using all the data and getting a close estimate of the error, but they're beyond the scope of this survey.

Ethics Obviously any time you deal with personal information, ethical considerations arise. Databases often include sensitive information that shouldn't be released. Social Security numbers are just one of several pieces of data that one can use to gain access to a person's identity, which you don't want to get around *too* much.

Less obviously, data miners also need to be careful to avoid discrimination. For example, a bank that intends to use data mining to determine whether to approve loans should think twice before including race or gender as one of the attributes given to the learning algorithm. Even zip codes should probably not go into the learning algorithm, as implicit discrimination can arise due to communities with a particularly high density of one race.

For these applications, the data mining practitioner should review the generated hypothesis to look for unethical or illegal discrimination. Algorithms that generate meaningful hypothesis (like linear regression or ID3, but not nearest-neighbor search) are particularly useful for such applications that need human review at the end.

1.4 Conclusion

We have seen a sample of data mining techniques — linear regression, nearest-neighbor search, and ID3 — and other issues that data mining practitioners must heed. The topic of data mining is rich and just now becoming a widely applied field. We could easily spend a complete semester studying it — it involves many interesting applications of mathematics to this goal of data analysis. I'd personally love to spend more time on it — but I also know what's to come, and it's every bit as intriguing!