# Contents

CHAPTER 1

# Notation and conventions

A dataset as a collection of $d$-tuples (a $d$-tuple is an ordered list of $d$ elements). Tuples differ from vectors, because we can always add and subtract vectors, but we cannot necessarily add or subtract tuples. There are always $N$ items in any dataset. There are always $d$ elements in each tuple in a dataset. The number of elements will be the same for every tuple in any given tuple. Sometimes we may not know the value of some elements in some tuples.

We use the same notation for a tuple and for a vector. Most of our data will be vectors. We write a vector in bold, so $\mathbf{x}$ could represent a vector or a tuple (the context will make it obvious which is intended).

The entire data set is $\{\mathbf{x}\}$. When we need to refer to the $i$'th data item, we write $\mathbf{x}_i$. Assume we have $N$ data items, and we wish to make a new dataset out of them; we write the dataset made out of these items as $\{\mathbf{x}_i\}$ (the $i$ is to suggest you are taking a set of items and making a dataset out of them). If we need to refer to the $j$'th component of a vector $\mathbf{x}_i$, we will write $x_i^{(j)}$ (notice this isn't in bold, because it is a component not a vector, and the $j$ is in parentheses because it isn't a power). Vectors are always column vectors.

**Terms:**

- mean $(\{x\})$ is the mean of the dataset $\{x\}$ (definition 1, page 22).
- std $(x)$ is the standard deviation of the dataset $\{x\}$ (definition 2, page 24).
- var $(\{x\})$ is the standard deviation of the dataset $\{x\}$ (definition 3, page 27).
- median $(\{x\})$ is the standard deviation of the dataset $\{x\}$ (definition 4, page 28).
- percentile$(\{x\}, k)$ is the $k\%$ percentile of the dataset $\{x\}$ (definition 5, page 29).
- iqr$\{x\}$ is the interquartile range of the dataset $\{x\}$ (definition 7, page 29).
- $\{\hat{x}\}$ is the dataset $\{x\}$, transformed to standard coordinates (definition 8, page 34).
- Standard normal data is defined in definition 9, page 35).
- Normal data is defined in definition 10, page 36).
- corr $(\{(x, y)\})$ is the correlation between two components $x$ and $y$ of a dataset (definition 11, page 46).
- $\emptyset$ is the empty set.
- $\Omega$ is the set of all possible outcomes of an experiment.
- Sets are written as $\mathcal{A}$.

- $\mathcal{A}^c$ is the complement of the set $\mathcal{A}$ (i.e. $\Omega - \mathcal{A}$).
- $\mathcal{E}$ is an event (page 114).
- $P(\{\mathcal{E}\})$ is the probability of event $\mathcal{E}$ (page 114).
- $P(\{\mathcal{E}\}|\{\mathcal{F}\})$ is the probability of event $\mathcal{E}$, conditioned on event $\mathcal{F}$ (page 114).
- $p(x)$ is the probability that random variable $X$ will take the value $x$; also written $P(\{X = x\})$ (page 114).
- $p(x, y)$ is the probability that random variable $X$ will take the value $x$ and random variable $Y$ will take the value $y$; also written $P(\{X = x\} \cap \{Y = y\})$ (page 114).
- $\underset{x}{\text{argmax}}\, f(x)$ means the value of $x$ that maximises $f(x)$.
- $\hat{\theta}$ is an estimated value of a parameter $\theta$.

**Background information:**

- *Cards:* A standard deck of playing cards contains 52 cards. These cards are divided into four suits. The suits are: spades and clubs (which are black); and hearts and diamonds (which are red). Each suit contains 13 cards: Ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack (sometimes called Knave), Queen and King. It is common to call Jack, Queen and King *court cards*.
- *Dice:* If you look hard enough, you can obtain dice with many different numbers of sides (though I've never seen a three sided die). We adopt the convention that the sides of an $N$ sided die are labeled with the numbers $1 \ldots N$, and that no number is used twice. Most dice are like this.
- *Fairness:* Each face of a fair coin or die has the same probability of landing upmost in a flip or roll.

## 1.1 SOME USEFUL MATHEMATICAL FACTS

The gamma function $\Gamma(x)$ is defined by a series of steps. First, we have that for $n$ an integer,

$$\Gamma(n) = (n-1)!$$

and then for $z$ a complex number with positive real part (which includes positive real numbers), we have

$$\Gamma(z) = \int_0^\infty t^z \frac{e^{-t}}{t} dt.$$

By doing this, we get a function on positive real numbers that is a smooth interpolate of the factorial function. We won't do any real work with this function, so won't expand on this definition. In practice, we'll either look up a value in tables or require a software environment to produce it.

## 1.2 ACKNOWLEDGEMENTS

Typos spotted by: Han Chen (numerous!), Yusuf Sobh, Scott Walters, Eric Huber, — Your Name Here —

CHAPTER 2

# First Tools for Looking at Data

The single most important question for a working scientist — perhaps the single most useful question anyone can ask — is: "what's going on here?" Answering this question requires creative use of different ways to make pictures of datasets, to summarize them, and to expose whatever structure might be there. This is an activity that is sometimes known as "Descriptive Statistics". There isn't any fixed recipe for understanding a dataset, but there is a rich variety of tools we can use to get insights.

## 2.1 DATASETS

A dataset is a collection of descriptions of different instances of the same phenomenon. These descriptions could take a variety of forms, but it is important that they are descriptions of the same thing. For example, my grandfather collected the daily rainfall in his garden for many years; we could collect the height of each person in a room; or the number of children in each family on a block; or whether 10 classmates would prefer to be "rich" or "famous". There could be more than one description recorded for each item. For example, when he recorded the contents of the rain gauge each morning, my grandfather could have recorded (say) the temperature and barometric pressure. As another example, one might record the height, weight, blood pressure and body temperature of every patient visiting a doctor's office.

The descriptions in a dataset can take a variety of forms. A description could be **categorical**, meaning that each data item can take a small set of prescribed values. For example, we might record whether each of 100 passers-by preferred to be "Rich" or "Famous". As another example, we could record whether the passers-by are "Male" or "Female". Categorical data could be **ordinal**, meaning that we can tell whether one data item is larger than another. For example, a dataset giving the number of children in a family for some set of families is categorical, because it uses only non-negative integers, but it is also ordinal, because we can tell whether one family is larger than another.

Some ordinal categorical data appears not to be numerical, but can be assigned a number in a reasonably sensible fashion. For example, many readers will recall being asked by a doctor to rate their pain on a scale of 1 to 10 — a question that is usually relatively easy to answer, but is quite strange when you think about it carefully. As another example, we could ask a set of users to rate the usability of an interface in a range from "very bad" to "very good", and then record that using -2 for "very bad", -1 for "bad", 0 for "neutral", 1 for "good", and 2 for "very good".

Many interesting datasets involve **continuous** variables (like, for example, height or weight or body temperature) when you could reasonably expect to encounter any value in a particular range. For example, we might have the heights of

all people in a particular room; or the rainfall at a particular place for each day of the year; or the number of children in each family on a list.

You should think of a dataset as a collection of $d$-tuples (a $d$-tuple is an ordered list of $d$ elements). Tuples differ from vectors, because we can always add and subtract vectors, but we cannot necessarily add or subtract tuples. We will always write $N$ for the number of tuples in the dataset, and $d$ for the number of elements in each tuple. The number of elements will be the same for every tuple, though sometimes we may not know the value of some elements in some tuples (which means we must figure out how to predict their values, which we will do much later).

| Index | net worth |
|---|---|
| 1 | 100, 360 |
| 2 | 109, 770 |
| 3 | 96, 860 |
| 4 | 97, 860 |
| 5 | 108, 930 |
| 6 | 124, 330 |
| 7 | 101, 300 |
| 8 | 112, 710 |
| 9 | 106, 740 |
| 10 | 120, 170 |

| Index | Taste score | Index | Taste score |
|---|---|---|---|
| 1 | 12.3 | 11 | 34.9 |
| 2 | 20.9 | 12 | 57.2 |
| 3 | 39 | 13 | 0.7 |
| 4 | 47.9 | 14 | 25.9 |
| 5 | 5.6 | 15 | 54.9 |
| 6 | 25.9 | 16 | 40.9 |
| 7 | 37.3 | 17 | 15.9 |
| 8 | 21.9 | 18 | 6.4 |
| 9 | 18.1 | 19 | 18 |
| 10 | 21 | 20 | 38.9 |

TABLE 2.1: *On the* **left**, *net worths of people you meet in a bar, in US $; I made this data up, using some information from the US Census. The index column, which tells you which data item is being referred to, is usually not displayed in a table because you can usually assume that the first line is the first item, and so on. On the* **right**, *the taste score (I'm not making this up; higher is better) for 20 different cheeses. This data is real (i.e. not made up), and it comes from* `http://lib.stat.cmu.edu/DASL/Datafiles/Cheese.html`.

Each element of a tuple has its own type. Some elements might be categorical. For example, one dataset we shall see several times records entries for Gender; Grade; Age; Race; Urban/Rural; School; Goals; Grades; Sports; Looks; and Money for 478 children, so $d = 11$ and $N = 478$. In this dataset, each entry is categorical data. Clearly, these tuples are not vectors because one cannot add or subtract (say) Genders.

Most of our data will be vectors. We use the same notation for a tuple and for a vector. We write a vector in bold, so $\mathbf{x}$ could represent a vector or a tuple (the context will make it obvious which is intended).

The entire data set is $\{\mathbf{x}\}$. When we need to refer to the $i$'th data item, we write $\mathbf{x}_i$. Assume we have $N$ data items, and we wish to make a new dataset out of them; we write the dataset made out of these items as $\{\mathbf{x}_i\}$ (the $i$ is to suggest you are taking a set of items and making a dataset out of them).

In this chapter, we will work mainly with continuous data. We will see a variety of methods for plotting and summarizing 1-tuples. We can build these plots from a dataset of $d$-tuples by extracting the $r$'th element of each $d$-tuple.

Mostly, we will deal with continuous data. All through the book, we will see many datasets downloaded from various web sources, because people are so generous about publishing interesting datasets on the web. In the next chapter, we will look at 2-dimensional data, and we look at high dimensional data in chapter 4.

## 2.2 WHAT'S HAPPENING? - PLOTTING DATA

The very simplest way to present or visualize a dataset is to produce a table. Tables can be helpful, but aren't much use for large datasets, because it is difficult to get any sense of what the data means from a table. As a continuous example, table 2.1 gives a table of the net worth of a set of people you might meet in a bar (I made this data up). You can scan the table and have a rough sense of what is going on; net worths are quite close to $ 100, 000, and there aren't any very big or very small numbers. This sort of information might be useful, for example, in choosing a bar.

People would like to measure, record, and reason about an extraordinary variety of phenomena. Apparently, one can score the goodness of the flavor of cheese with a number (bigger is better); table 2.1 gives a score for each of thirty cheeses (I did not make up this data, but downloaded it from `http://lib.stat.cmu.edu/DASL/Datafiles/Cheese.html`). You should notice that a few cheeses have very high scores, and most have moderate scores. It's difficult to draw more significant conclusions from the table, though.

| Gender | Goal | Gender | Goal |
|---|---|---|---|
| boy | Sports | girl | Sports |
| boy | Popular | girl | Grades |
| girl | Popular | boy | Popular |
| girl | Popular | boy | Popular |
| girl | Popular | boy | Popular |
| girl | Popular | girl | Grades |
| girl | Popular | girl | Sports |
| girl | Grades | girl | Popular |
| girl | Sports | girl | Grades |
| girl | Sports | girl | Sports |

TABLE 2.2: *Chase and Dunner (?) collected data on what students thought made other students popular. As part of this effort, they collected information on (a) the gender and (b) the goal of students. This table gives the gender ("boy" or "girl") and the goal (to make good grades —"Grades"; to be popular — "Popular"; or to be good at sports — "Sports"). The table gives this information for the first 20 of 478 students; the rest can be found at* `http://lib.stat.cmu.edu/DASL/Datafiles/PopularKids.html`. *This data is clearly categorical, and not ordinal.*

Table 2.2 shows a table for a set of categorical data. Psychologists collected data from students in grades 4-6 in three school districts to understand what factors students thought made other students popular. This fascinating data set can be found at `http://lib.stat.cmu.edu/DASL/Datafiles/PopularKids.html`, and was prepared by Chase and Dunner (**?**). Among other things, for each student

they asked whether the student's goal was to make good grades ("Grades", for short); to be popular ("Popular"); or to be good at sports ("Sports"). They have this information for 478 students, so a table would be very hard to read. Table 2.2 shows the gender and the goal for the first 20 students in this group. It's rather harder to draw any serious conclusion from this data, because the full table would be so big. We need a more effective tool than eyeballing the table.

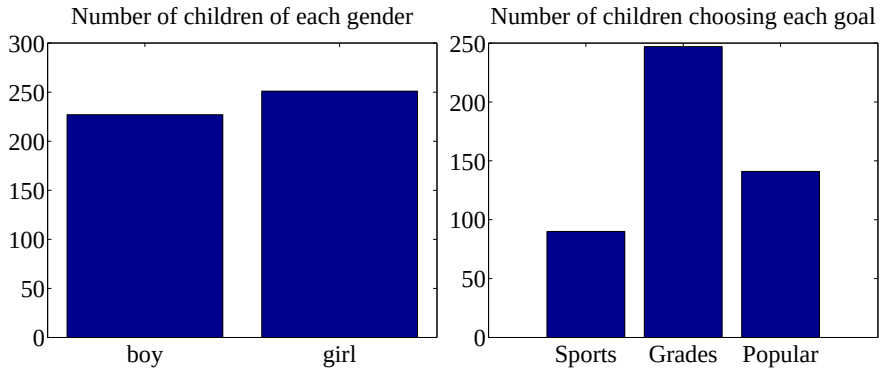


FIGURE 2.1: *On the* **left**, *a bar chart of the number of children of each gender in the Chase and Dunner study (). Notice that there are about the same number of boys and girls (the bars are about the same height). On the* **right**, *a bar chart of the number of children selecting each of three goals. You can tell, at a glance, that different goals are more or less popular by looking at the height of the bars.*

### 2.2.1 Bar Charts

A **bar chart** is a set of bars, one per category, where the height of each bar is proportional to the number of items in that category. A glance at a bar chart often exposes important structure in data, for example, which categories are common, and which are rare. Bar charts are particularly useful for categorical data. Figure 2.1 shows such bar charts for the genders and the goals in the student dataset of Chase and Dunner (). You can see at a glance that there are about as many boys as girls, and that there are more students who think grades are important than students who think sports or popularity is important. You couldn't draw either conclusion from Table 2.2, because I showed only the first 20 items; but a 478 item table is very difficult to read.

### 2.2.2 Histograms

Data is continuous when a data item could take any value in some range or set of ranges. In turn, this means that we can reasonably expect a continuous dataset contains few or no pairs of items that have *exactly* the same value. Drawing a bar chart in the obvious way — one bar per value — produces a mess of unit height bars, and seldom leads to a good plot. Instead, we would like to have fewer bars, each representing more data items. We need a procedure to decide which data items count in which bar.

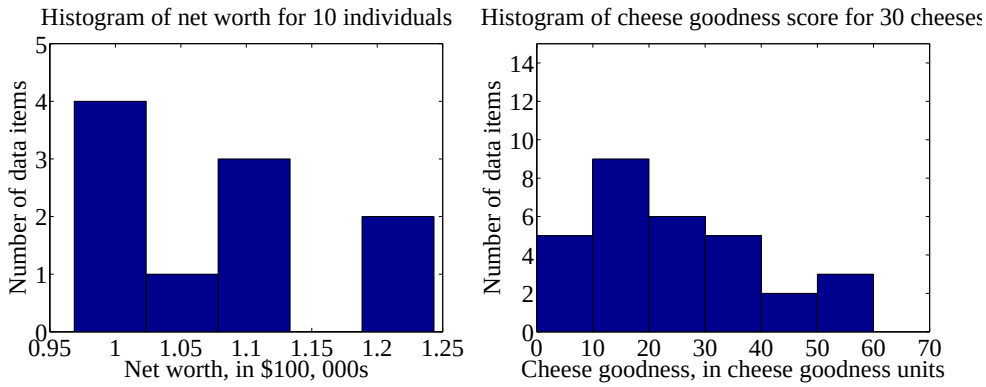


FIGURE 2.2: *On the* **left**, *a histogram of net worths from the dataset described in the text and shown in table 2.1. On the* **right**, *a histogram of cheese goodness scores from the dataset described in the text and shown in table 2.1.*

A simple generalization of a bar chart is a **histogram**. We divide the range of the data into intervals, which do not need to be equal in length. We think of each interval as having an associated pigeonhole, and choose one pigeonhole for each data item. We then build a set of boxes, one per interval. Each box sits on its interval on the horizontal axis, and its height is determined by the number of data items in the corresponding pigeonhole. In the simplest histogram, the intervals that form the bases of the boxes are equally sized. In this case, the height of the box is given by the number of data items in the box.

Figure 2.2 shows a histogram of the data in table 2.1. There are five bars — by my choice; I could have plotted ten bars — and the height of each bar gives the number of data items that fall into its interval. For example, there is one net worth in the range between $102, 500 and $107, 500. Notice that one bar is invisible, because there is no data in that range. This picture suggests conclusions consistent with the ones we had from eyeballing the table — the net worths tend to be quite similar, and around $100, 000.

Figure 2.2 shows a histogram of the data in table 2.1. There are six bars (0-10, 10-20, and so on), and the height of each bar gives the number of data items that fall into its interval — so that, for example, there are 9 cheeses in this dataset whose score is greater than or equal to 10 and less than 20. You can also use the bars to estimate other properties. So, for example, there are 14 cheeses whose score is less than 20, and 3 cheeses with a score of 50 or greater. This picture is much more helpful than the table; you can see at a glance that quite a lot of cheeses have relatively low scores, and few have high scores.

### 2.2.3 Conditional Histograms

Most people believe that normal body temperature is $98.4^o$ in Fahrenheit. If you take other people's temperatures often (for example, you might have children), you know that some individuals tend to run a little warmer or a little cooler than this number. I found data giving the body temperature of a set of individuals at

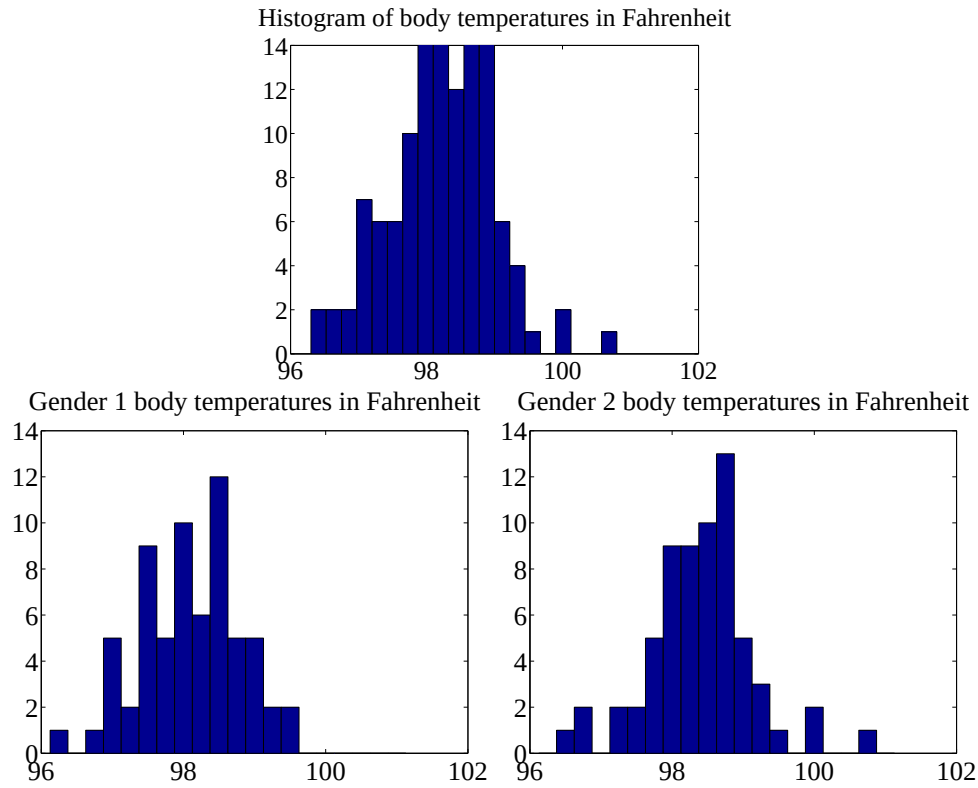


FIGURE 2.3: *On* **top**, *a histogram of body temperatures, from the dataset published at* `http://www2.stetson.edu/~jrasp/data.htm`. *These seem to be clustered fairly tightly around one value. The* **bottom row** *shows histograms for each gender (I don't know which is which). It looks as though one gender runs slightly cooler than the other.*

`http://www2.stetson.edu/~jrasp/data.htm`. As you can see from the histogram (figure 2.3), the body temperatures cluster around a small set of numbers. But what causes the variation?

One possibility is gender. We can investigate this possibility by comparing a histogram of temperatures for males with histogram of temperatures for females. Such histograms are sometimes called **conditional histograms** or **class-conditional histograms**, because each histogram is conditioned on something (in this case, the histogram uses only data that comes from gender).

The dataset gives genders (as 1 or 2 - I don't know which is male and which female). Figure 2.3 gives the class conditional histograms. It does seem like individuals of one gender run a little cooler than individuals of the other, although we don't yet have mechanisms to test this possibility in detail (chapter 1).

## 2.3 PLOTTING 2D DATA

We take a dataset, choose two different entries, and extract the corresponding elements from each tuple. The result is a dataset consisting of 2-tuples, and we think of this as a two dimensional dataset. The first step is to plot this dataset in a way that reveals relationships. The topic of how best to plot data fills many books, and we can only scratch the surface here. Categorical data can be particularly tricky, because there are a variety of choices we can make, and the usefulness of each tends to depend on the dataset and to some extent on one's cleverness in graphic design (section 2.3.1).

For some continuous data, we can plot the one entry as a function of the other (so, for example, our tuples might consist of the date and the number of robberies; or the year and the price of lynx pelts; and so on, section 2.3.2).

Mostly, we use a simple device, called a scatter plot. Using and thinking about scatter plots will reveal a great deal about the relationships between our data items (section 2.3.3).

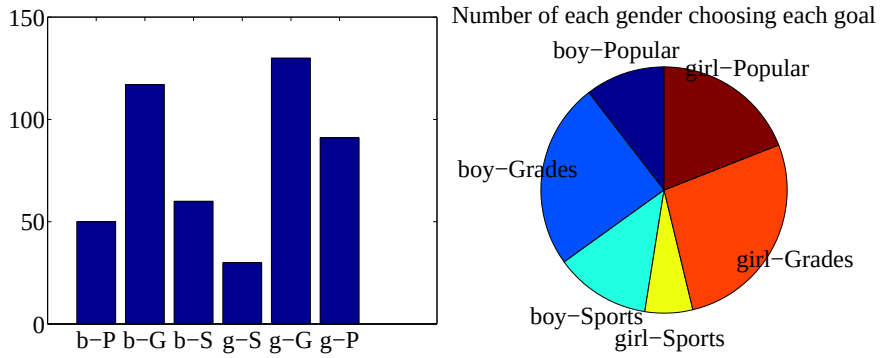


FIGURE 2.4: *I sorted the children in the Chase and Dunner study into six categories (two genders by three goals), and counted the number of children that fell into each cell. I then produced the bar chart on the* **left**, *which shows the number of children of each gender, selecting each goal. On the* **right**, *a pie chart of this information. I have organized the pie chart so it is easy to compare boys and girls by eye — start at the top; going down on the left side are boy goals, and on the right side are girl goals. Comparing the size of the corresponding wedges allows you to tell what goals boys (resp. girls) identify with more or less often.*

### 2.3.1 Categorical Data, Counts, and Charts

Categorical data is a bit special. Assume we have a dataset with several categorical descriptions of each data item. One way to plot this data is to think of it as belonging to a richer set of categories. Assume the dataset has categorical descriptions, which are not ordinal. Then we can construct a new set of categories by looking at each of the cases for each of the descriptions. For example, in the Chase and Dunner data of table 2.2, our new categories would be: "boy-sports"; "girl-sports"; "boy-popular"; "girl-popular"; "boy-grades"; and "girl-grades". A large set of cat-

egories like this can result in a poor bar chart, though, because there may be too many bars to group the bars successfully. Figure 2.4 shows such a bar chart. Notice that it is hard to group categories by eye to compare; for example, you can see that slightly more girls think grades are important than boys do, but to do so you need to compare two bars that are separated by two other bars. An alternative is a **pie chart**, where a circle is divided into sections whose angle is proportional to the size of the data item. You can think of the circle as a pie, and each section as a slice of pie. Figure 2.4 shows a pie chart, where each section is proportional to the number of students in its category. In this case, I've used my judgement to lay the categories out in a way that makes comparisons easy. I'm not aware of any tight algorithm for doing this, though.

Pie charts have problems, because it is hard to judge small differences in area accurately by eye. For example, from the pie chart in figure 2.4, it's hard to tell that the "boy-sports" category is slightly bigger than the "boy-popular" category (try it; check using the bar chart). For either kind of chart, it is quite important to think about *what* you plot. For example, the plot of figure 2.4 shows the total number of respondents, and if you refer to figure 2.1, you will notice that there are slightly more girls in the study. Is the *percentage* of boys who think grades are important smaller (or larger) than the *percentage* of girls who think so? you can't tell from these plots, and you'd have to plot the percentages instead.

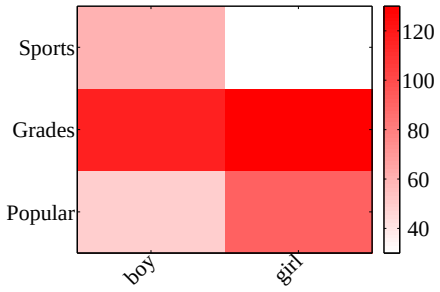


FIGURE 2.5: *A heat map of the Chase and Dunner data. The color of each cell corresponds to the count of the number of elements of that type. The colorbar at the side gives the correspondence between color and count. You can see at a glance that the number of boys and girls who prefer grades is about the same; that about the same number of boys prefer sports and popularity, with sports showing a mild lead; and that more girls prefer popularity to sports.*

An alternative to a pie chart that is very useful for two dimensional data is a **heat map**. This is a method of displaying a matrix as an image. Each entry of the matrix is mapped to a color, and the matrix is represented as an image. For the Chase and Dunner study, I constructed a matrix where each row corresponds to a choice of "sports", "grades", or "popular", and each column corresponds to a choice of "boy" or "girl". Each entry contains the count of data items of that type. Zero values are represented as white; the largest values as red; and as the value increases, we use an increasingly saturated pink. This plot is shown in figure 2.5

If the categorical data is ordinal, the ordering offers some hints for making

| | -2 | -1 | 0 | 1 | 2 |
|---|---|---|---|---|---|
| -2 | 24 | 5 | 0 | 0 | 1 |
| -1 | 6 | 12 | 3 | 0 | 0 |
| 0 | 2 | 4 | 13 | 6 | 0 |
| 1 | 0 | 0 | 3 | 13 | 2 |
| 2 | 0 | 0 | 0 | 1 | 5 |

TABLE 2.3: *I simulated data representing user evaluations of a user interface. Each cell in the table on the* **left** *contains the count of users rating "ease of use" (horizontal, on a scale of -2 -very bad- to 2 -very good) vs. "enjoyability" (vertical, same scale). Users who found the interface hard to use did not like using it either. While this data is categorical, it's also ordinal, so that the order of the cells is determined. It wouldn't make sense, for example, to reorder the columns of the table or the rows of the table.*

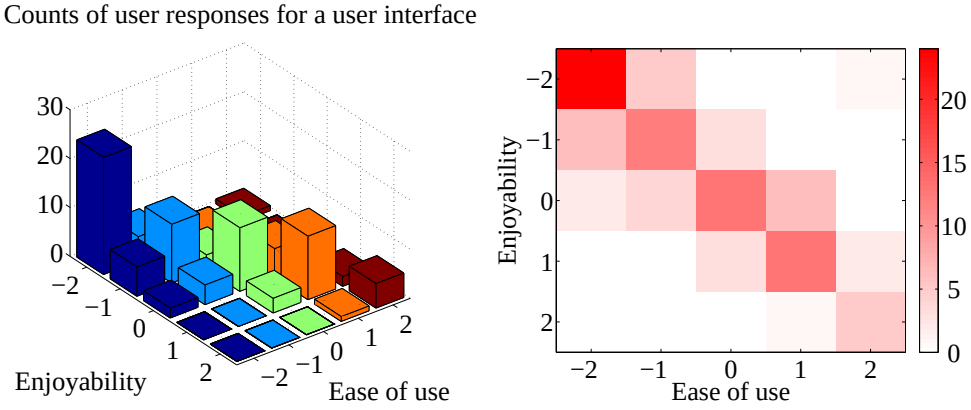


FIGURE 2.6: *On the* **left**, *a 3D bar chart of the data. The height of each bar is given by the number of users in each cell. This figure immediately reveals that users who found the interface hard to use did not like using it either. However, some of the bars at the back are hidden, so some structure might be hard to infer. On the* **right**, *a heat map of this data. Again, this figure immediately reveals that users who found the interface hard to use did not like using it either. It's more apparent that everyone disliked the interface, though, and it's clear that there is no important hidden structure.*

a good plot. For example, imagine we are building a user interface. We build an initial version, and collect some users, asking each to rate the interface on scales for "ease of use" (-2, -1, 0, 1, 2, running from bad to good) and "enjoyability" (again, -2, -1, 0, 1, 2, running from bad to good). It is natural to build a 5x5 table, where each cell represents a pair of "ease of use" and "enjoyability" values. We then count the number of users in each cell, and build graphical representations of this table. One natural representation is a **3D bar chart**, where each bar sits on its cell in the 2D table, and the height of the bars is given by the number of elements in the cell. Table 2.3 shows a table and figure 2.6 shows a 3D bar chart for some simulated

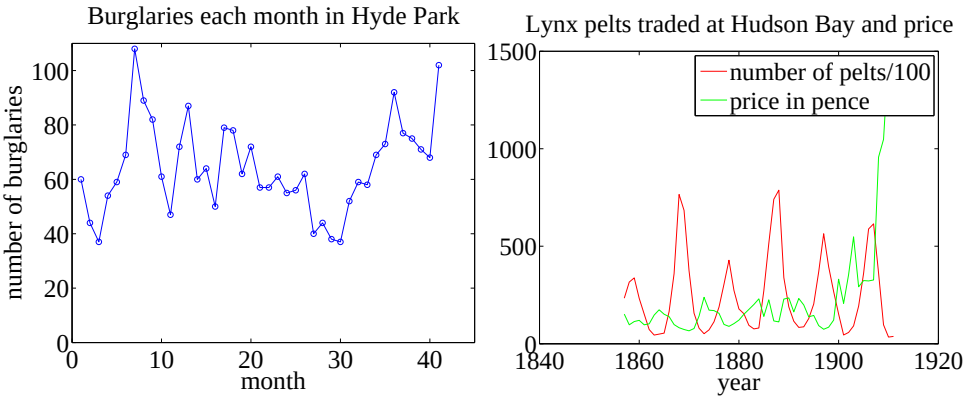


FIGURE 2.7: **Left**, *the number of burglaries in Hyde Park, by month.* **Right**, *a plot of the number of lynx pelts traded at Hudson Bay and of the price paid per pelt, as a function of the year. Notice the scale, and the legend box (the number of pelts is scaled by 100).*

data. The main difficulty with a 3D bar chart is that some bars are hidden behind others. This is a regular nuisance. You can improve things by using an interactive tool to rotate the chart to get a nice view, but this doesn't always work. Heatmaps don't suffer from this problem (Figure 2.6), another reason they are a good choice.

### 2.3.2 Series

Sometimes one component of a dataset gives a natural ordering to the data. For example, we might have a dataset giving the maximum rainfall for each day of the year. We could record this either by using a two-dimensional representation, where one dimension is the number of the day and the other is the temperature, or with a convention where the $i$'th data item is the rainfall on the $i$'th day. For example, at `http://lib.stat.cmu.edu/DASL/Datafiles/timeseriesdat.html`, you can find four datasets indexed in this way. It is natural to plot data like this as a function of time. From this dataset, I extracted data giving the number of burglaries each month in a Chicago suburb, Hyde Park. I have plotted part this data in Figure 2.7 (I left out the data to do with treatment effects). It is natural to plot a graph of the burglaries as a function of time (in this case, the number of the month). The plot shows each data point explicitly. I also told the plotting software to draw lines joining data points, because burglaries do not all happen on a specific day. The lines suggest, reasonably enough, the rate at which burglaries are happening between data points.

As another example, at `http://lib.stat.cmu.edu/datasets/Andrews/` you can find a dataset that records the number of lynx pelts traded to the Hudson's Bay company and the price paid for each pelt. This version of the dataset appeared first in table 3.2 of *Data: a Collection of Problems from many Fields for the Student and Research Worker* by D.F. Andrews and A.M. Herzberg, published by Springer in 1985. I have plotted it in figure 2.7. The dataset is famous, because it shows

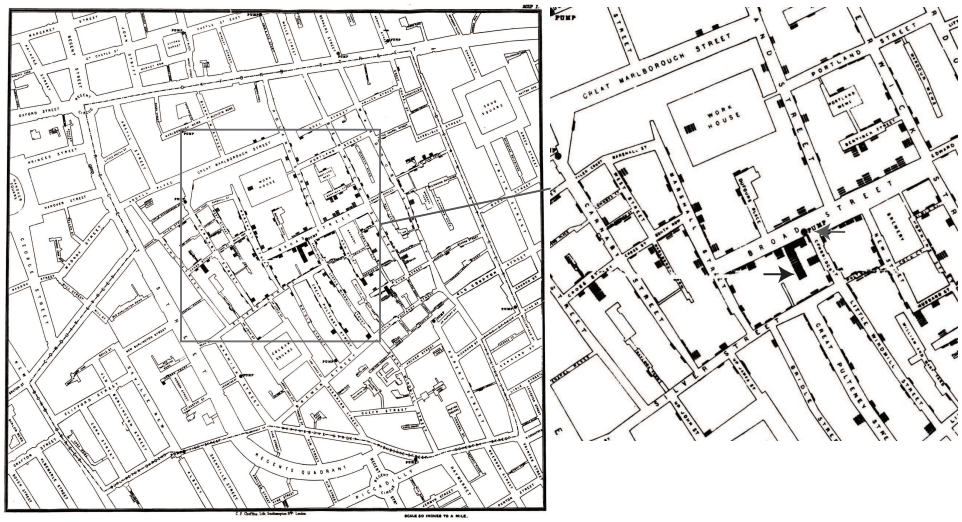


FIGURE 2.8: *Snow's scatter plot of cholera deaths on the* **left**. *Each cholera death is plotted as a small bar on the house in which the bar occurred (for example, the black arrow points to one stack of these bars, indicating many deaths, in the detail on the* **right***). Notice the fairly clear pattern of many deaths close to the Broad street pump (grey arrow in the detail), and fewer deaths further away (where it was harder to get water from the pump).*

a periodic behavior in the number of pelts (which is a good proxy for the number of lynx), which is interpreted as a result of predator-prey interactions. Lynx eat rabbits. When there are many rabbits, lynx kittens thrive, and soon there will be many lynx; but then they eat most of the rabbits, and starve, at which point the rabbit population rockets. You should also notice that after about 1900, prices seem to have gone up rather quickly. I don't know why this is. There is also some suggestion, as there should be, that prices are low when there are many pelts, and high when there are few.

### 2.3.3 Scatter Plots for Spatial Data

It isn't always natural to plot data as a function. For example, in a dataset containing the temperature and blood pressure of a set of patients, there is no reason to believe that temperature is a function of blood pressure, or the other way round. Two people could have the same temperature, and different blood pressures, or vice-versa. As another example, we could be interested in what causes people to die of cholera. We have data indicating *where* each person died in a particular outbreak. It isn't helpful to try and plot such data as a function.

The **scatter plot** is a powerful way to deal with this situation. In the first instance, assume that our data points actually describe points on the a real map. Then, to make a scatter plot, we make a mark on the map at a place indicated by each data point. What the mark looks like, and how we place it, depends on the

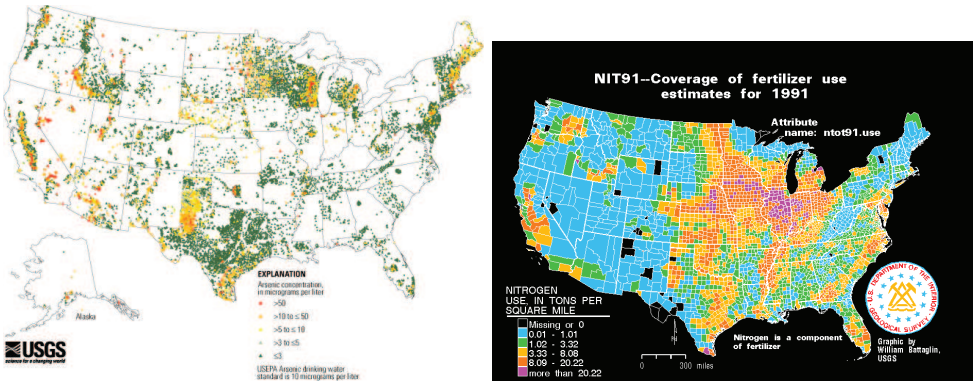


FIGURE 2.9: **Left**, *a scatter plot of arsenic levels in US groundwater, prepared by the US Geological Survey (you can find the data at* `http://water.usgs.gov/GIS/metadata/usgswrd/XML/arsenic_map.xml`*. Here the shape and color of each marker shows the amount of arsenic, and the spatial distribution of the markers shows where the wells were sampled.* **Right**, *the usage of Nitrogen (a component of fertilizer) by US county in 1991, prepared by the US Geological Survey (you can find the data at* `http://water.usgs.gov/GIS/metadata/usgswrd/XML/nit91.xml`*). In this variant of a scatter plot (which usually takes specialized software to prepare) one fills each region with a color indicating the data in that region.*

particular dataset, what we are looking for, how much we are willing to work with complex tools, and our sense of graphic design.

Figure 2.8 is an extremely famous scatter plot, due to John Snow. Snow — one of the founders of epidemiology — used a scatter plot to reason about a cholera outbreak centered on the Broad Street pump in London in 1854. At that time, the mechanism that causes cholera was not known. Snow plotted cholera deaths as little bars (more bars, more deaths) on the location of the house where the death occurred. More bars means more deaths, fewer bars means fewer deaths. There are more bars per block close to the pump, and few far away. This plot offers quite strong evidence of an association between the pump and death from cholera. Snow used this scatter plot as evidence that cholera was associated with water, and that the Broad Street pump was the source of the tainted water.

Figure 2.9 shows a scatter plot of arsenic levels in groundwater for the United States, prepared by the US Geological Survey. The data set was collected by Focazio and others in 2000; by Welch and others in 2000; and then updated by Ryker 2001. It can be found at `http://water.usgs.gov/GIS/metadata/usgswrd/XML/arsenic_map.xml`. One variant of a scatter plot that is particularly useful for geographic data occurs when one fills regions on a map with different colors, following the data in that region. Figure 2.9 shows the nitrogen usage by US county in 1991; again, this figure was prepared by the US Geological Survey.

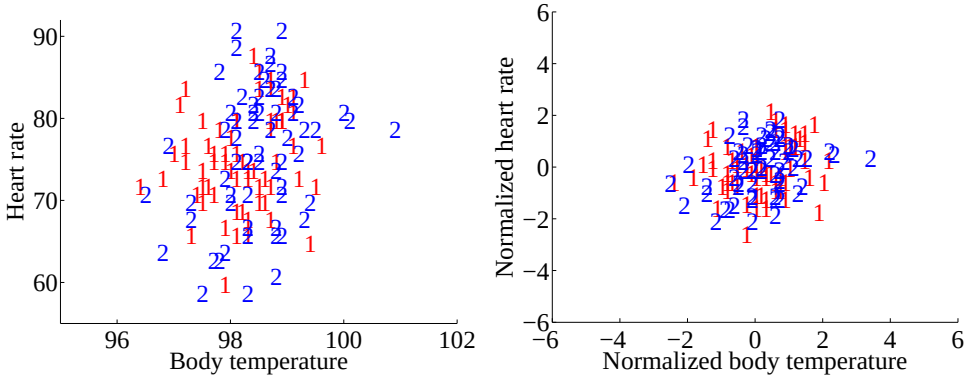


FIGURE 2.10: *A scatter plot of body temperature against heart rate, from the dataset at* `http://www2.stetson.edu/~jrasp/data.htm`*; normtemp.xls. I have separated the two genders by plotting a different symbol for each (though I don't know which gender is indicated by which letter); if you view this in color, the differences in color makes for a greater separation of the scatter. This picture suggests, but doesn't conclusively establish, that there isn't much dependence between temperature and heart rate, and any dependence between temperature and heart rate isn't affected by gender.*

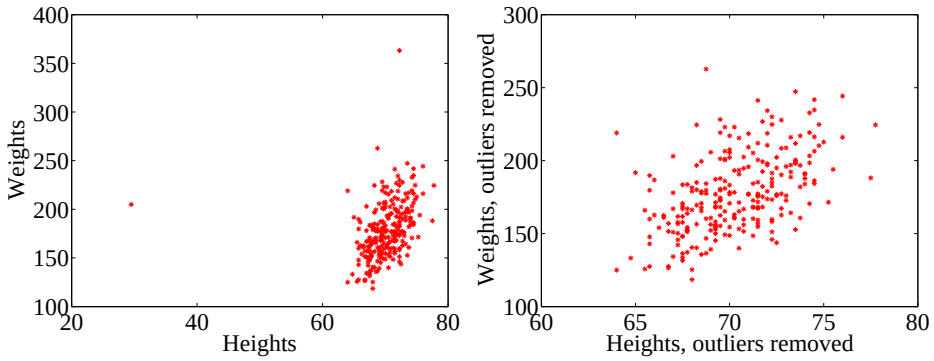


FIGURE 2.11: *A scatter plots of weight against height, from the dataset at* `http://www2.stetson.edu/~jrasp/data.htm`*.* **Left:** *Notice how two outliers dominate the picture, and to show the outliers, the rest of the data has had to be bunched up.* **Right** *shows the data with the outliers removed. The structure is now somewhat clearer.*

### 2.3.4 Scatter Plots — Scale is a problem

Scatter plots are natural for geographic data, but a scatter plot is a useful, simple tool for ferreting out associations in other kinds of data as well. Now we need some notation. Assume we have a dataset $\{x\}$ of $N$ data items, $x_1, \ldots, x_N$. Each data item is a $d$ dimensional vector (so its components are numbers). We wish to investigate the relationship between two components of the dataset. For example,

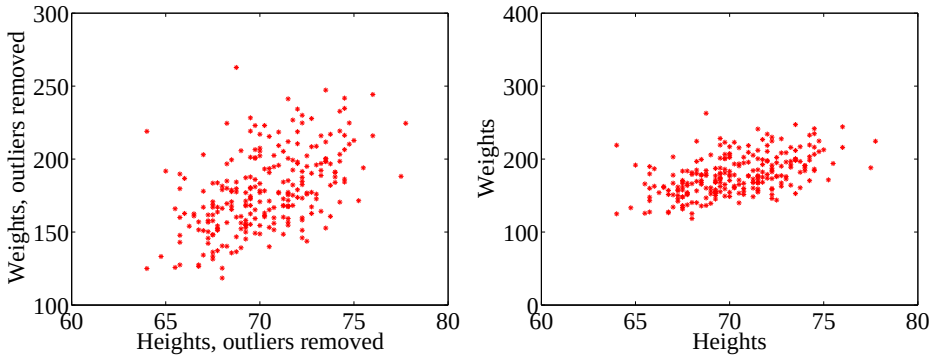


FIGURE 2.12: *Scatter plots of weight against height, from the dataset at* `http://www2.stetson.edu/~jrasp/data.htm`. **Left:** *data with two outliers removed, as in figure 2.11.* **Right:** *this data, rescaled slightly. Notice how the data looks less spread out. But there is no difference between the datasets. Instead, your eye is easily confused by a change of scale.*

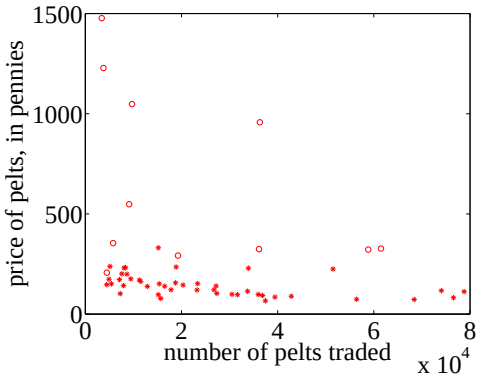


FIGURE 2.13: *A scatter plot of the price of lynx pelts against the number of pelts. I have plotted data for 1901 to the end of the series as circles, and the rest of the data as *'s. It is quite hard to draw any conclusion from this data, because the scale is confusing. Furthermore, the data from 1900 on behaves quite differently from the other data.*

we might be interested in the 7'th and the 13'th component of the dataset. We will produce a two-dimensional plot, one dimension for each component. It does not really matter which component is plotted on the $x$-coordinate and which on the $y$-coordinate (though it will be some pages before this is clear). But it is very difficult to write sensibly without talking about the $x$ and $y$ coordinates.

We will make a two-dimensional dataset out of the components that interest us. We must choose which component goes first in the resulting 2-vector. We will plot this component on the $x$-coordinate (and we refer to it as the $x$-coordinate), and to the other component as the $y$-coordinate. This is just to make it easier to

describe what is going on; there's no important idea here. It really will not matter which is $x$ and which is $y$. The two components make a dataset $\{\mathbf{x}_i\} = \{(x_i, y_i)\}$. To produce a scatter plot of this data, we plot a small shape at the location of each data item.

Such scatter plots are very revealing. For example, figure 2.10 shows a scatter plot of body temperature against heart rate for humans. In this dataset, the gender of the subject was recorded (as "1" or "2" — I don't know which is which), and so I have plotted a "1" at each data point with gender "1", and so on. Looking at the data suggests there isn't much difference between the blob of "1" labels and the blob of "2" labels, which suggests that females and males are about the same in this respect.

The scale used for a scatter plot matters. For example, plotting lengths in meters gives a very different scatter from plotting lengths in millimeters. Figure 2.11 shows two scatter plots of weight against height. Each plot is from the same dataset, but one is scaled so as to show two outliers. Keeping these outliers means that the rest of the data looks quite concentrated, just because the axes are in large units. In the other plot, the axis scale has changed (so you can't see the outliers), but the data looks more scattered. This may or may not be a misrepresentation. Figure 2.12 compares the data with outliers removed, with the same plot on a somewhat different set of axes. One plot looks as though increasing height corresponds to increasing weight; the other looks as though it doesn't. This is purely due to deceptive scaling — each plot shows the same dataset.

Dubious data can also contribute to scaling problems. Recall that, in figure 2.7, price data before and after 1900 appeared to behave differently. Figure 2.13 shows a scatter plot of the lynx data, where I have plotted number of pelts against price. I plotted the post-1900 data as circles, and the rest as asterisks. Notice how the circles seem to form a quite different figure, which supports the suggestion that something interesting happened around 1900. The scatter plot does not seem to support the idea that prices go up when supply goes down, which is puzzling, because this is a pretty reliable idea. This turns out to be a scale effect. Scale is an important nuisance, and it's easy to get misled by scale effects.

CHAPTER 3

# Summaries and Plots

## 3.1 SUMMARIZING 1D DATA

For the rest of this chapter, we will assume that data items take values that are continuous real numbers. Furthermore, we will assume that values can be added, subtracted, and multiplied by constants in a meaningful way. Human heights are one example of such data; you can add two heights, and interpret the result as a height (perhaps one person is standing on the head of the other). You can subtract one height from another, and the result is meaningful. You can multiply a height by a constant — say, $1/2$ — and interpret the result (A is half as high as B). Not all data is like this. Categorical data is often not like this. For example, you could not add "Grades" to "Popular" in any useful way.

### 3.1.1 The Mean

One simple and effective summary of a set of data is its **mean**. This is sometimes known as the **average** of the data.

**Definition: 3.1** *Mean*

Assume we have a dataset $\{x\}$ of $N$ data items, $x_1, \ldots, x_N$. Their mean is

$$\mathsf{mean}\left(\{x\}\right) = \frac{1}{N}\sum_{i=1}^{i=N} x_i.$$

For example, assume you're in a bar, in a group of ten people who like to talk about money. They're average people, and their net worth is given in table 2.1 (you can choose who you want to be in this story). The mean of this data is \$107, 903.

An important interpretation of the mean is that it is the best guess of the value of a new data item, given no information at all. In the bar example, if a new person walked into this bar, and you had to guess that person's net worth, you should choose \$107, 903.

**Properties of the Mean** The mean has several important properties you should remember:

- Scaling data scales the mean: or $\mathsf{mean}\left(\{kx_i\}\right) = k\mathsf{mean}\left(\{x_i\}\right)$.
- Translating data translates the mean: or $\mathsf{mean}\left(\{x_i + c\}\right) = \mathsf{mean}\left(\{x_i\}\right) + c$.

- The sum of signed differences from the mean is zero. This means that

$$\sum_{i=1}^{N}(x_i - \text{mean}\,(\{x_i\})) = 0.$$

- Choose the number $\mu$ such that the sum of squared distances of data points to $\mu$ is minimized. That number is the mean. In notation

$$\underset{\mu}{\arg\min} \sum_i (x_i - \mu)^2 = \text{mean}\,(\{x_i\})$$

These properties are easy to prove (and so easy to remember). All but one proof is relegated to the exercises.

**Proposition:** $\underset{\mu}{arg\ min} \sum_i (x_i - \mu)^2 = \text{mean}\,(\{x\})$

**Proof:** Choose the number $\mu$ such that the sum of squared distances of data points to $\mu$ is minimized. That number is the mean. In notation:

$$\underset{\mu}{\arg\min} \sum_i (x_i - \mu)^2 = \text{mean}\,(\{x\})$$

We can show this by actually minimizing the expression. We must have that the derivative of the expression we are minimizing is zero at the value of $\mu$ we are seeking. So we have

$$\begin{aligned}\frac{d}{d\mu}\sum_{i=1}^{N}(x_i - \mu)^2 &= \sum_{i=1}^{N} 2(x_i - \mu) \\ &= 2\sum_{i=1}^{N}(x_i - \mu) \\ &= 0\end{aligned}$$

so that $2N\text{mean}\,(\{x\}) - 2N\mu = 0$, which means that $\mu = \text{mean}\,(\{x\})$.

**Property 3.1:** The Average Squared Distance to the Mean is Minimized

### 3.1.2 Standard Deviation and Variance

We would also like to know the extent to which data items are close to the mean. This information is given by the **standard deviation**, which is the root mean square of the offsets of data from the mean.

**Definition: 3.2** *Standard deviation*

Assume we have a dataset $\{x\}$ of $N$ data items, $x_1, \ldots, x_N$. The standard deviation of this dataset is is:

$$\mathsf{std}\left(x_i\right) = \sqrt{\frac{1}{N}\sum_{i-1}^{i=N}(x_i - \mathsf{mean}\left(\{x\}\right))^2} = \sqrt{\mathsf{mean}\left(\{(x_i - \mathsf{mean}\left(\{x\}\right))^2\}\right)}.$$

You should think of the standard deviation as a scale. It measures the size of the average deviation from the mean for a dataset. When the standard deviation of a dataset is large, there are many items with values much larger than, or much smaller than, the mean. When the standard deviation is small, most data items have values close to the mean. This means it is helpful to talk about how many standard devations away from the mean a particular data item is. Saying that data item $x_j$ is "within $k$ standard deviations from the mean" means that

$$\mathsf{abs}\left(x_j - \mathsf{mean}\left(\{x\}\right)\right) \leq k\mathsf{std}\left(x_i\right).$$

Similarly, saying that data item $x_j$ is "more than $k$ standard deviations from the mean" means that

$$\mathsf{abs}\left(x_i - \mathsf{mean}\left(\{x\}\right)\right) > k\mathsf{std}\left(x\right).$$

As I will show below, there must be some data at least one standard deviation away from the mean, and there can be very few data items that are many standard deviations away from the mean.

**Properties of the Standard Deviation** Standard deviation has very important properties:

- Translating data does not change the standard deviation, i.e. $\mathsf{std}\left(x_i + c\right) = \mathsf{std}\left(x_i\right)$.
- Scaling data scales the standard deviation, i.e. $\mathsf{std}\left(kx_i\right) = k\mathsf{std}\left(x_i\right)$.
- For any dataset, there can be only a few items that are many standard deviations away from the mean. In particular, assume we have $N$ data items, $x_i$, whose standard deviation is $\sigma$. Then there are at most $\frac{1}{k^2}$ data points lying $k$ or more standard deviations away from the mean.
- For any dataset, there must be at least one data item that is at least one standard deviation away from the mean.

The first two properties are easy to prove, and are relegated to the exercises.

**Proposition:** *Assume we have a dataset $\{x\}$ of $N$ data items, $x_1, \ldots, x_N$. Assume the standard deviation of this dataset is $\mathsf{std}(x) = \sigma$. Then there are at most $\frac{1}{k^2}$ data points lying $k$ or more standard deviations away from the mean.*

**Proof:** Assume the mean is zero. There is no loss of generality here, because translating data translates the mean, but doesn't change the standard deviation. The way to prove this is to construct a dataset with the largest possible fraction $r$ of data points lying $k$ or more standard deviations from the mean. To achieve this, our data should have $N(1-r)$ data points each with the value 0, because these contribute 0 to the standard deviation. It should have $Nr$ data points with the value $k\sigma$; if they are further from zero than this, each will contribute more to the standard deviation, so the fraction of such points will be fewer. Because

$$\mathsf{std}(x) = \sigma = \sqrt{\frac{\sum_i x_i^2}{N}}$$

we have that, for this rather specially constructed dataset,

$$\sigma = \sqrt{\frac{Nrk^2\sigma^2}{N}}$$

so that

$$r = \frac{1}{k^2}.$$

We constructed the dataset so that $r$ would be as large as possible, so

$$r \geq \frac{1}{k^2}$$

*for any kind of data at all.*

**Property 3.2:** For any dataset, it is hard for data items to get many standard deviations away from the mean.

The bound of box 3.1.2 is true for *any kind of data*. This bound implies that, for example, at most 100% of *any* dataset could be one standard deviation away from the mean, 25% of *any* dataset is 2 standard deviations away from the mean and at most 11% of *any* dataset could be 3 standard deviations away from the mean. But the configuration of data that achieves this bound is very unusual. This means the bound tends to wildly overstate how much data is far from the mean for most practical datasets. Most data has more random structure, meaning that we expect to see very much *less* data far from the mean than the bound predicts. For example, much data can reasonably be modelled as coming from a normal distribution (a topic we'll go into later). For such data, we expect that about 68% of the data is within one standard deviation of the mean, 95% is within two standard deviations of the mean, and 99.7% is within three standard deviations of the mean, and the percentage of data that is within ten standard deviations of the mean is essentially indistinguishable from 100%. This kind of behavior is quite common; the crucial point about the standard deviation is that you won't see much

data that lies many standard deviations from the mean, because you can't.

**Proposition:** $(\mathsf{std}\,(x))^2 \leq \max_i (x_i - \mathsf{mean}\,(\{x\}))^2$.

**Proof:** You can see this by looking at the expression for standard deviation. We have

$$\mathsf{std}\,(x) = \sqrt{\frac{1}{N}\sum_{i-1}^{i=N}(x_i - \mathsf{mean}\,(\{x\}))^2}.$$

Now, this means that

$$N(\mathsf{std}\,(x))^2 = \sum_{i-1}^{i=N}(x_i - \mathsf{mean}\,(\{x\}))^2.$$

But

$$\sum_{i-1}^{i=N}(x_i - \mathsf{mean}\,(\{x\}))^2 \leq N \max_i (x_i - \mathsf{mean}\,(\{x\}))^2$$

so

$$(\mathsf{std}\,(x))^2 \leq \max_i (x_i - \mathsf{mean}\,(\{x\}))^2.$$

**Property 3.3:** For any dataset, there must be at least one data item that is at least one standard deviation away from the mean.

Boxes 3.1.2 and 3.1.2 mean that the standard deviation is quite informative. Very little data is many standard deviations away from the mean; similarly, at least some of the data should be one or more standard deviations away from the mean. So the standard deviation tells us how data points are scattered about the mean.

**Potential point of confusion:** There is an ambiguity that comes up often here because two (very slightly) different numbers are called the standard deviation of a dataset. One — the one we use in this chapter — is an estimate of the scale of the data, as we describe it. The other differs from our expression very slightly; one computes

$$\sqrt{\frac{\sum_i (x_i - \mathsf{mean}\,(\{x\}))^2}{N-1}}$$

(notice the $N-1$ for our $N$). If $N$ is large, this number is basically the same as the number we compute, but for smaller $N$ there is a difference that can be significant. Irritatingly, this number is also called the standard deviation; even more irritatingly, we will have to deal with it, but not yet. I mention it now because you may look up terms I have used, find this definition, and wonder. Don't worry - the $N$ in our expressions is the right thing to use for what we're doing.

### 3.1.3 Variance

It turns out that thinking in terms of the square of the standard deviation, which is known as the **variance**, will allow us to generalize our summaries to apply to higher dimensional data.

**Definition: 3.3** *Variance*

Assume we have a dataset $\{x\}$ of $N$ data items, $x_1, \ldots, x_N$. where $N > 1$. Their variance is:

$$\mathsf{var}\left(\{x\}\right) = \frac{1}{N}\left(\sum_{i-1}^{i=N}(x_i - \mathsf{mean}\left(\{x\}\right))^2\right) = \mathsf{mean}\left(\left\{(x_i - \mathsf{mean}\left(\{x\}\right))^2\right\}\right).$$

One good way to think of the variance is as the mean-square error you would incur if you replaced each data item with the mean. Another is that it is the square of the standard deviation.

**Properties of the Variance** The properties of the variance follow from the fact that it is the square of the standard deviation. We have that:

- Translating data does not change the variance, i.e. $\mathsf{var}\left(\{x + c\}\right) = \mathsf{var}\left(\{x\}\right)$.
- Scaling data scales the variance by a square of the scale, i.e. $\mathsf{var}\left(\{kx\}\right) = k^2\mathsf{var}\left(\{x\}\right)$.

While one could restate the other two properties of the standard deviation in terms of the variance, it isn't really natural to do so. The standard deviation is in the same units as the original data, and should be thought of as a scale. Because the variance is the square of the standard deviation, it isn't a natural scale (unless you take its square root!).

### 3.1.4 The Median

One problem with the mean is that it can be affected strongly by extreme values. Go back to the bar example, of section 3.1.1. Now Warren Buffett (or Bill Gates, or your favorite billionaire) walks in. What happened to the average net worth?

Assume your billionaire has net worth \$ 1, 000, 000, 000. Then the mean net worth suddenly has become

$$\frac{10 \times \$107,903 + \$1,000,000,000}{11} = \$91,007,184$$

But this mean isn't a very helpful summary of the people in the bar. It is probably more useful to think of the net worth data as ten people together with one billionaire. The billionaire is known as an **outlier**.

One way to get outliers is that a small number of data items are very different, due to minor effects you don't want to model. Another is that the data was misrecorded, or mistranscribed. Another possibility is that there is just too much variation in the data to summarize it well. For example, a small number of extremely wealthy people could change the average net worth of US residents

dramatically, as the example shows. An alternative to using a mean is to use a **median**.

**Definition: 3.4** *Median*

The median of a set of data points is obtained by sorting the data points, and finding the point halfway along the list. If the list is of even length, it's usual to average the two numbers on either side of the middle. We write

$$\mathsf{median}\left(\{x_i\}\right)$$

for the operator that returns the median.

For example,

$$\mathsf{median}\left(\{3,5,7\}\right) = 5,$$

$$\mathsf{median}\left(\{3,4,5,6,7\}\right) = 5,$$

and

$$\mathsf{median}\left(\{3,4,5,6\}\right) = 4.5.$$

For much, but not all, data, you can expect that roughly half the data is smaller than the median, and roughly half is larger than the median. Sometimes this property fails. For example,

$$\mathsf{median}\left(\{1,2,2,2,2,2,2,2,3\}\right) = 2.$$

With this definition, the median of our list of net worths is $\$107,835$. If we insert the billionaire, the median becomes $\$108,930$. Notice by how little the number has changed — it remains an effective summary of the data.

**Properties of the median** You can think of the median of a dataset as giving the "middle" or "center" value. This means it is rather like the mean, which also gives a (slightly differently defined) "middle" or "center" value. The mean has the important properties that if you translate the dataset, the mean translates, and if you scale the dataset, the mean scales. The median has these properties, too:

- Translating data translates the median, i.e. $\mathsf{median}\left(\{x+c\}\right) = \mathsf{median}\left(\{x\}\right) + c$.
- Scaling data scales the median by the same scale, i.e. $\mathsf{median}\left(\{kx\}\right) = k\mathsf{median}\left(\{x\}\right)$.

Each is easily proved, and proofs are relegated to the exercises.

### 3.1.5 Interquartile Range

Outliers can affect standard deviations severely, too. For our net worth data, the standard deviation without the billionaire is $\$9265$, but if we put the billionaire in there, it is $\$3.014 \times 10^8$. When the billionaire is in the dataset, all but one of

the data items lie about a third of a standard deviation away from the mean; the other one (the billionaire) is many standard deviations away from the mean. In this case, the standard deviation has done its work of informing us that there are huge changes in the data, but isn't really helpful.

The problem is this: describing the net worth data with billionaire as a having a mean of $\$9.101 \times 10^7$ with a standard deviation of $\$3.014 \times 10^8$ really isn't terribly helpful. Instead, the data really should be seen as a clump of values that are near $\$100,000$ and moderately close to one another, and one massive number (the billionaire outlier).

One thing we could do is simply remove the billionaire and compute mean and standard deviation. This isn't always easy to do, because it's often less obvious which points are outliers. An alternative is to follow the strategy we did when we used the median. Find a summary that describes scale, but is less affected by outliers than the standard deviation. This is the **interquartile range**; to define it, we need to define percentiles and quartiles, which are useful anyway.

**Definition: 3.5** *Percentile*

The $k$'th percentile is the value such that $k\%$ of the data is less than or equal to that value. We write $\mathsf{percentile}(\{x\}, k)$ for the $k$'th percentile of dataset $\{x\}$.

**Definition: 3.6** *Quartiles*

The first quartile of the data is the value such that 25% of the data is less than or equal to that value (i.e. $\mathsf{percentile}(\{x\}, 25)$). The second quartile of the data is the value such that 50% of the data is less than or equal to that value, which is usually the median (i.e. $\mathsf{percentile}(\{x\}, 50)$). The third quartile of the data is the value such that 75% of the data is less than or equal to that value (i.e. $\mathsf{percentile}(\{x\}, 75)$).

**Definition: 3.7** *Interquartile Range*

The interquartile range of a dataset $\{x\}$ is $\mathsf{iqr}\{x\} = \mathsf{percentile}(\{x\}, 75) - \mathsf{percentile}(\{x\}, 25)$.

Like the standard deviation, the interquartile range gives an estimate of how widely the data is spread out. But it is quite well-behaved in the presence of outliers. For our net worth data without the billionaire, the interquartile range is \$12350; with the billionaire, it is \$17710.

**Properties of the interquartile range** You can think of the interquartile range of a dataset as giving an estimate of the scale of the difference from the mean. This means it is rather like the standard deviation, which also gives a (slightly differently defined) scale. The standard deviation has the important properties that if you translate the dataset, the standard deviation translates, and if you scale the dataset, the standard deviation scales. The interquartile range has these properties, too:

- Translating data does not change the interquartile range, i.e. $\mathsf{iqr}\{x + c\} = \mathsf{iqr}\{x\}$.
- Scaling data scales the interquartile range by the same scale, i.e. $\mathsf{iqr}\{kx\} = k^2\mathsf{iqr}\{x\}$.

Each is easily proved, and proofs are relegated to the exercises.

### 3.1.6 Using Summaries Sensibly

One should be careful how one summarizes data. For example, the statement that "the average US family has 2.6 children" invites mockery (the example is from Andrew Vickers' book *What is a p-value anyway?*), because you can't have fractions of a child — no family has 2.6 children. A more accurate way to say things might be "the average of the number of children in a US family is 2.6", but this is clumsy. What is going wrong here is the 2.6 is a mean, but the number of children in a family is a categorical variable. Reporting the mean of a categorical variable is often a bad idea, because you may never encounter this value (the 2.6 children). For a categorical variable, giving the median value and perhaps the interquartile range often makes much more sense than reporting the mean.

For continuous variables, reporting the mean is reasonable because you could expect to encounter a data item with this value, even if you haven't seen one in the particular data set you have. It is sensible to look at both mean and median; if they're significantly different, then there is probably something going on that is worth understanding. You'd want to plot the data using the methods of the next section before you decided what to report.

You should also be careful about how precisely numbers are reported (equivalently, the number of significant figures). Numerical and statistical software will produce very large numbers of digits freely, but not all are always useful. This is a particular nuisance in the case of the mean, because you might add many numbers, then divide by a large number; in this case, you will get many digits, but some might not be meaningful. For example, Vickers (ibid) describes a paper reporting the mean length of pregnancy as 32.833 weeks. That fifth digit suggests we know the mean length of pregnancy to about 0.001 weeks, or roughly 10 minutes. Neither medical interviewing nor people's memory for past events is that detailed. Furthermore, when you interview them about embarrassing topics, people quite often lie. There is no prospect of knowing this number with this precision.

People regularly report silly numbers of digits because it is easy to miss the harm caused by doing so. But the harm is there: you are implying to other people, and to yourself, that you know something more accurately than you do. At some point, someone will suffer for it.

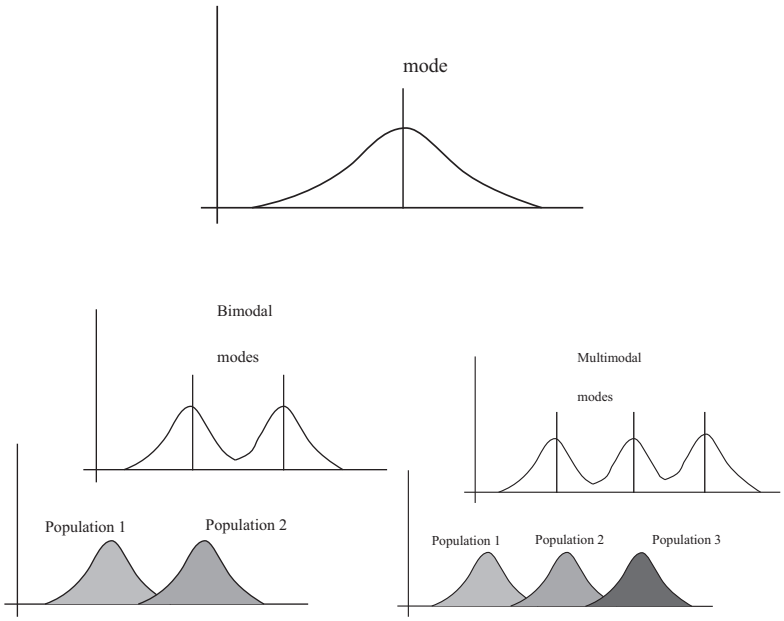


FIGURE 3.1: *Many histograms are unimodal, like the example on the* **top***; there is one peak, or mode. Some are bimodal (two peaks;* **bottom left***) or even multimodal (two or more peaks;* **bottom right***). One common reason (but not the only reason) is that there are actually two populations being conflated in the histograms. For example, measuring adult heights might result in a bimodal histogram, if male and female heights were slightly different. As another example, measuring the weight of dogs might result in a multimodal histogram if you did not distinguish between breeds (eg chihauhau, terrier, german shepherd, pyranean mountain dog, etc.).*

## 3.2 PLOTS AND SUMMARIES

Knowing the mean, standard deviation, median and interquartile range of a dataset gives us some information about what its histogram might look like. In fact, the summaries give us a language in which to describe a variety of characteristic properties of histograms that are worth knowing about (Section 3.2.1). Quite remarkably, many different datasets have the same shape of histogram (Section 3.2.2). For such data, we know roughly what percentage of data items are how far from the mean.

Complex datasets can be difficult to interpret with histograms alone, because it is hard to compare many histograms by eye. Section 3.2.3 describes a clever plot of various summaries of datasets that makes it easier to compare many cases.

### 3.2.1 Some Properties of Histograms

The **tails** of a histogram are the relatively uncommon values that are significantly larger (resp. smaller) than the value at the peak (which is sometimes called the **mode**). A histogram is **unimodal** if there is only one peak; if there are more than one, it is **multimodal**, with the special term **bimodal** sometimes being used for

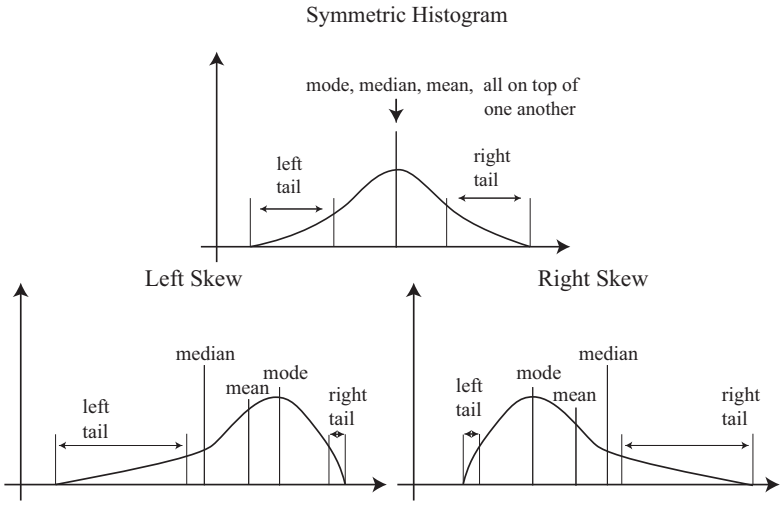


FIGURE 3.2: *On the* **top**, *an example of a symmetric histogram, showing its tails (relatively uncommon values that are significantly larger or smaller than the peak or mode).* **Lower left**, *a sketch of a left-skewed histogram. Here there are few large values, but some very small values that occur with significant frequency. We say the left tail is "long", and that the histogram is left skewed (confusingly, this means the main bump is to the right).* **Lower right**, *a sketch of a right-skewed histogram. Here there are few small values, but some very large values that occur with significant frequency. We say the right tail is "long", and that the histogram is right skewed (confusingly, this means the main bump is to the left).*

the case where there are two peaks (Figure 3.1). The histograms we have seen have been relatively symmetric, where the left and right tails are about as long as one another. Another way to think about this is that values a lot larger than the mean are about as common as values a lot smaller than the mean. Not all data is symmetric. In some datasets, one or another tail is longer (figure 3.2). This effect is called **skew**.

Skew appears often in real data. SOCR (the Statistics Online Computational Resource) publishes a number of datasets. Here we discuss a dataset of citations to faculty publications. For each of five UCLA faculty members, SOCR collected the number of times each of the papers they had authored had been cited by other authors (data at `http://wiki.stat.ucla.edu/socr/index.php/SOCR_Data_Dinov_072108_H_Index_Pubs`). Generally, a small number of papers get many citations, and many papers get few citations. We see this pattern in the histograms of citation numbers (figure 3.3). These are very different from (say) the body temperature pictures. In the citation histograms, there are many data items that have very few citations, and few that have many citations. This means that the right tail of the histogram is longer, so the histogram is skewed to the right.

One way to check for skewness is to look at the histogram; another is to compare mean and median (though this is not foolproof). For the first citation

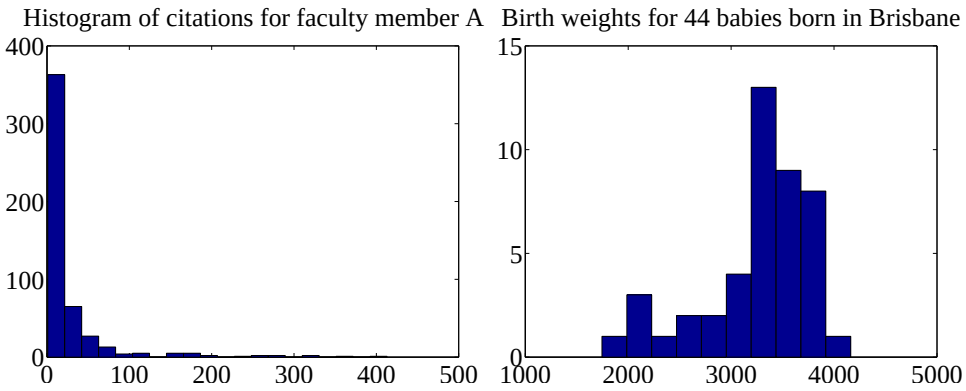


FIGURE 3.3: *On the* **left**, *a histogram of citations for a faculty member, from data at* `http://wiki.stat.ucla.edu/socr/index.php/SOCR_Data_Dinov_072108_H_Index_Pubs`. *Very few publications have many citations, and many publications have few. This means the histogram is strongly right-skewed. On the* **right**, *a histogram of birth weights for 44 babies borne in Brisbane in 1997. This histogram looks slightly left-skewed.*

histogram, the mean is 24.7 and the median is 7.5; for the second, the mean is 24.4, and the median is 11. In each case, the mean is a lot bigger than the median. Recall the definition of the median (form a ranked list of the data points, and find the point halfway along the list). For much data, the result is larger than about half of the data set and smaller than about half the dataset. So if the median is quite small compared to the mean, then there are many small data items and a small number of data items that are large — the right tail is longer, so the histogram is skewed to the right.

Left-skewed data also occurs; figure 3.3 shows a histogram of the birth weights of 44 babies born in Brisbane, in 1997 (from `http://www.amstat.org/publications/jse/jse_data_archive.htm`). This data appears to be somewhat left-skewed, as birth weights can be a lot smaller than the mean, but tend not to be much larger than the mean.

Skewed data is often, but not always, the result of constraints. For example, good obstetrical practice tries to ensure that very large birth weights are rare (birth is typically induced before the baby gets too heavy), but it may be quite hard to avoid some small birth weights. This could could skew birth weights to the left (because large babies will get born, but will not be as heavy as they could be if obstetricians had not interfered). Similarly, income data can be skewed to the right by the fact that income is always positive. Test mark data is often skewed — whether to right or left depends on the circumstances — by the fact that there is a largest possible mark and a smallest possible mark.

### 3.2.2 Standard Coordinates and Normal Data

It is useful to look at lots of histograms, because it is often possible to get some useful insights about data. However, in their current form, histograms are hard to

compare. This is because each is in a different set of units. A histogram for length data will consist of boxes whose horizontal units are, say, metres; a histogram for mass data will consist of boxes whose horizontal units are in, say, kilograms. Furthermore, these histograms typically span different ranges.

We can make histograms comparable by (a) estimating the "location" of the plot on the horizontal axis and (b) estimating the "scale" of the plot. The location is given by the mean, and the scale by the standard deviation. We could then normalize the data by subtracting the location (mean) and dividing by the standard deviation (scale). The resulting values are unitless, and have zero mean. They are often known as **standard coordinates**.

**Definition: 3.8** *Standard coordinates*

Assume we have a dataset $\{x\}$ of $N$ data items, $x_1, \ldots, x_N$. We represent these data items in standard coordinates by computing

$$\hat{x}_i = \frac{(x_i - \mathsf{mean}\,(\{x\}))}{\mathsf{std}\,(x)}.$$

We write $\{\hat{x}\}$ for a dataset that happens to be in standard coordinates.

Standard coordinates have some important properties. Assume we have $N$ data items. Write $x_i$ for the $i$'th data item, and $\hat{x}_i$ for the $i$'th data item in standard coordinates (I sometimes refer to these as "normalized data items"). Then we have

$$\mathsf{mean}\,(\{\hat{x}\}) = 0.$$

We also have that

$$\mathsf{std}\,(\hat{x}) = 1.$$

An extremely important fact about data is that, for many kinds of data, histograms of these standard coordinates look the same. Many completely different datasets produce a histogram that, in standard coordinates, has a very specific appearance. It is symmetric, unimodal; it looks like a narrow bump. If there were enough data points and the histogram boxes were small enough, the curve would look like the curve in figure 3.4. This phenomenon is so important that data of this form has a special name.

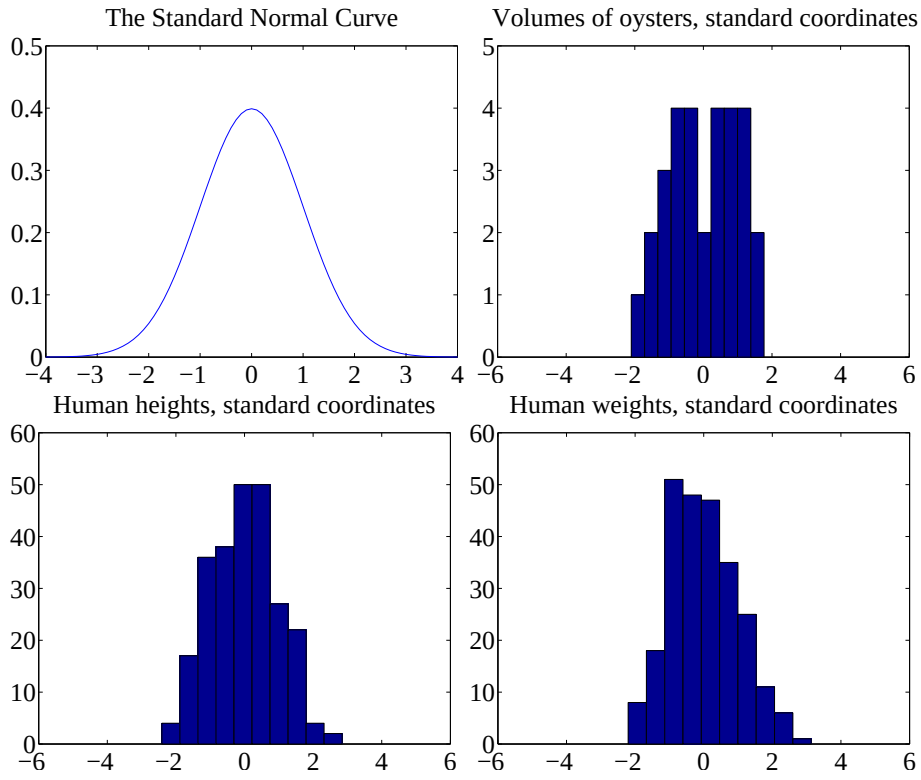


FIGURE 3.4: *Data is standard normal data when its histogram takes a stylized, bell-shaped form, plotted above. One usually requires a lot of data and very small histogram boxes for this form to be reproduced closely. Nonetheless, the histogram for normal data is unimodal (has a single bump) and is symmetric; the tails fall off fairly fast, and there are few data items that are many standard deviations from the mean. Many quite different data sets have histograms that are similar to the normal curve; I show three such datasets here.*

**Definition: 3.9** *Standard normal data*

Data is **standard normal data** if, when we have a great deal of data, the histogram is a close approximation to the **standard normal curve**. This curve is given by

$$y(x) = \frac{1}{\sqrt{2pi}} e^{\left(-x^2/2\right)}$$

(which is shown in figure 3.4).

**Definition: 3.10** *Normal data*

Data is **normal data** if, when we subtract the mean and divide by the standard deviation (i.e. compute standard coordinates), it becomes standard normal data.

It is not always easy to tell whether data is normal or not, and there are a variety of tests one can use, which we discuss later. However, there are many examples of normal data. Figure 3.4 shows a diverse variety of data sets, plotted as histograms in standard coordinates. These include: the volumes of 30 oysters (from `http://www.amstat.org/publications/jse/jse_data_archive.htm`; look for 30oysters.dat.txt); human heights (from `http://www2.stetson.edu/~jrasp/data.htm`; look for bodyfat.xls, with two outliers removed); and human weights (from `http://www2.stetson.edu/~jrasp/data.htm`; look for bodyfat.xls, with two outliers removed).

**Properties of normal data** For the moment, assume we know that a dataset is normal. Then we expect it to have the following properties:

- If we normalize it, its histogram will be close to the standard normal curve. This means, among other things, that the data is not significantly skewed.
- About 68% of the data lie within one standard deviation of the mean. We will prove this later.
- About 95% of the data lie within two standard deviations of the mean. We will prove this later.
- About 99% of the data lie within three standard deviations of the mean. We will prove this later.

In turn, these properties imply that data that contains outliers (points many standard deviations away from the mean) is not normal. This is usually a very safe assumption. It is quite common to model a dataset by excluding a small number of outliers, then modelling the remaining data as normal. For example, if I exclude two outliers from the height and weight data from `http://www2.stetson.edu/~jrasp/data.htm`, the data looks pretty close to normal.

### 3.2.3 Boxplots

It is usually hard to compare multiple histograms by eye. One problem with comparing histograms is the amount of space they take up on a plot, because each histogram involves multiple vertical bars. This means it is hard to plot multiple overlapping histograms cleanly. If you plot each one on a separate figure, you have to handle a large number of separate figures; either you print them too small to see enough detail, or you have to keep flipping over pages.

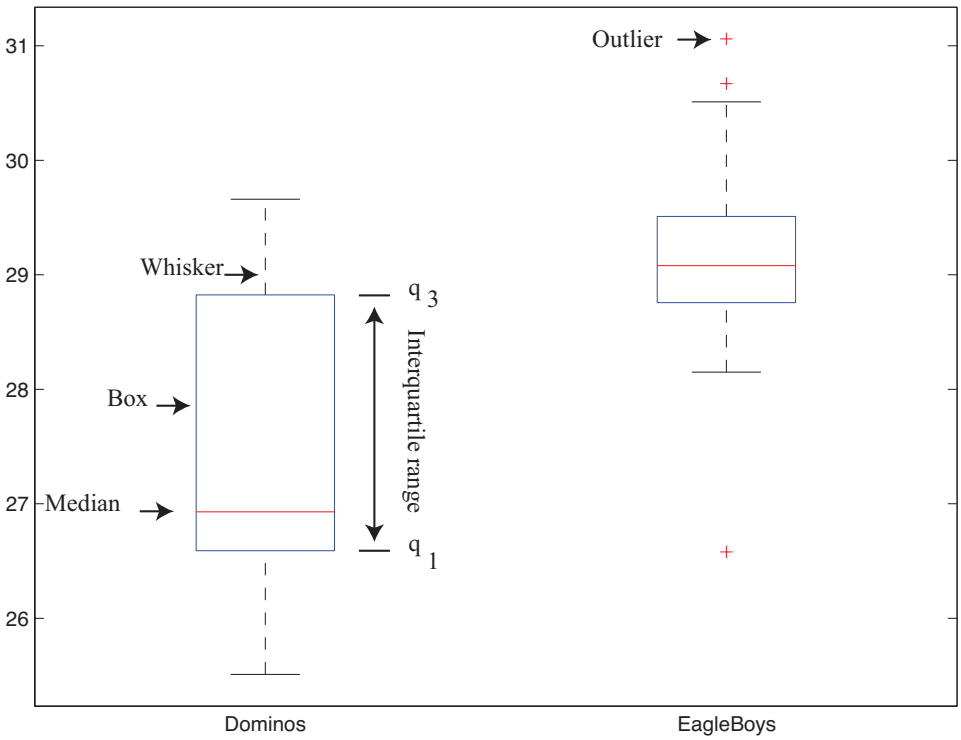


FIGURE 3.5: *A boxplot showing the box, the median, the whiskers and two outliers. Notice that we can compare the two datasets rather easily; the next section explains the comparison.*

A **boxplot** is a way to plot data that simplifies comparison. A boxplot displays a dataset as a vertical picture. There is a vertical box whose height corresponds to the interquartile range of the data (the width is just to make the figure easy to interpret). Then there is a horizontal line for the median; and the behavior of the rest of the data is indicated with whiskers and/or outlier markers. This means that each dataset makes is represented by a vertical structure, making it easy to show multiple datasets on one plot *and interpret the plot* (Figure 3.5).

To build a boxplot, we first plot a box that runs from the first to the third quartile. We then show the median with a horizontal line. We then decide which data items should be outliers. A variety of rules are possible; for the plots I show, I used the rule that data items that are larger than $q_3 + 1.5(q_3 - q_1)$ or smaller than $q_1 - 1.5(q_3 - q_1)$, are outliers. This criterion looks for data items that are more than one and a half interquartile ranges above the third quartile, or more than one and a half interquartile ranges below the first quartile.

Once we have identified outliers, we plot these with a special symbol (crosses in the plots I show). We then plot whiskers, which show the range of non-outlier data. We draw a whisker from $q_1$ to the smallest data item that is not an outlier, and from $q_3$ to the largest data item that is not an outlier. While all this sounds

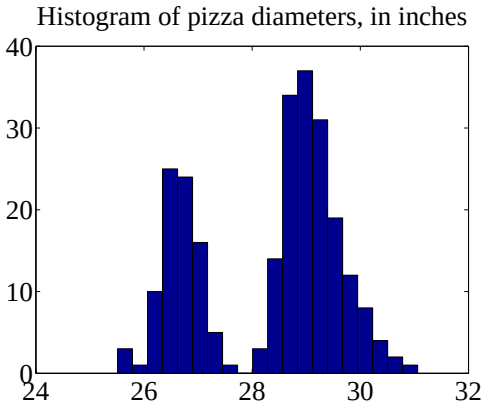


FIGURE 3.6: *A histogram of pizza diameters from the dataset described in the text. Notice that there seem to be two populations.*

complicated, any reasonable programming environment will have a function that will do it for you. Figure 3.5 shows an example boxplot. Notice that the rich graphical structure means it is quite straightforward to compare two histograms.

## 3.3 WHOSE IS BIGGER? INVESTIGATING AUSTRALIAN PIZZAS

At `http://www.amstat.org/publications/jse/jse_data_archive.htm`), you will find a dataset giving the diameter of pizzas, measured in Australia (search for the word "pizza"). This website also gives the backstory for this dataset. Apparently, EagleBoys pizza claims that their pizzas are always bigger than Dominos pizzas, and published a set of measurements to support this claim (the measurements were available at `http://www.eagleboys.com.au/realsizepizza` as of Feb 2012, but seem not to be there anymore).

Whose pizzas are bigger? and why? A histogram of all the pizza sizes appears in figure 3.6. We would not expect every pizza produced by a restaurant to have exactly the same diameter, but the diameters are probably pretty close to one another, and pretty close to some standard value. This would suggest that we'd expect to see a histogram which looks like a single, rather narrow, bump about a mean. This is not what we see in figure 3.6 — instead, there are two bumps, which suggests two populations of pizzas. This isn't particularly surprising, because we know that some pizzas come from EagleBoys and some from Dominos.

If you look more closely at the data in the dataset, you will notice that each data item is tagged with the company it comes from. We can now easily plot conditional histograms, conditioning on the company that the pizza came from. These appear in figure 3.7. Notice that EagleBoys pizzas seem to follow the pattern we expect — the diameters are clustered tightly around one value — but Dominos pizzas do not seem to be like that. This is reflected in a boxplot (figure 3.8), which shows the range of Dominos pizza sizes is surprisingly large, and that EagleBoys pizza sizes have several large outliers. There is more to understand about this data. The dataset contains labels for the type of crust and the type of topping — perhaps

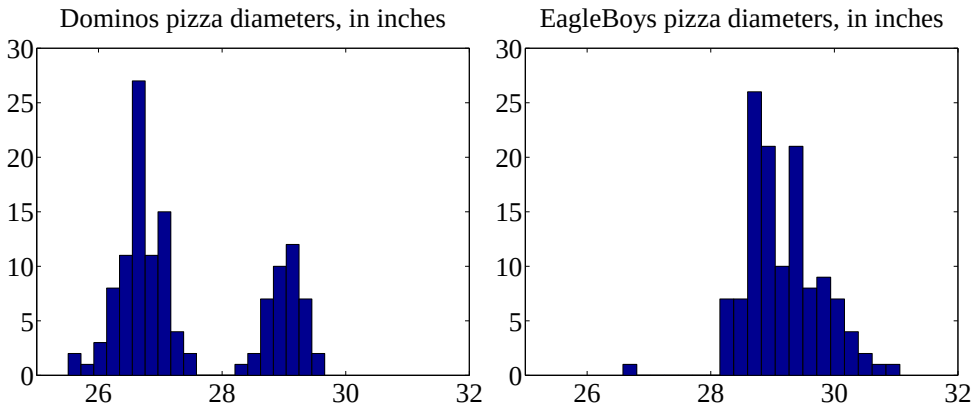


FIGURE 3.7: *On the* **left**, *the class-conditional histogram of Dominos pizza diameters from the pizza data set; on the* **right**, *the class-conditional histogram of EagleBoys pizza diameters. Notice that EagleBoys pizzas seem to follow the pattern we expect — the diameters are clustered tightly around a mean, and there is a small standard deviation — but Dominos pizzas do not seem to be like that. There is more to understand about this data.*

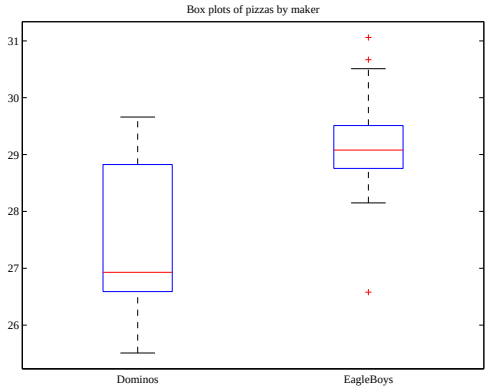


FIGURE 3.8: *Boxplots of the pizza data, comparing EagleBoys and Dominos pizza. There are several curiosities here: why is the range for Dominos so large (25.5-29)? EagleBoys has a smaller range, but has several substantial outliers; why? One would expect pizza manufacturers to try and control diameter fairly closely, because pizzas that are too small present risks (annoying customers; publicity; hostile advertising) and pizzas that are too large should affect profits.*

these properties affect the size of the pizza?

EagleBoys produces DeepPan, MidCrust and ThinCrust pizzas, and Dominos produces DeepPan, ClassicCrust and ThinNCrispy pizzas. This may have something to do with the observed patterns, but comparing six histograms by eye is unattractive. A boxplot is the right way to compare these cases (figure 3.9). The

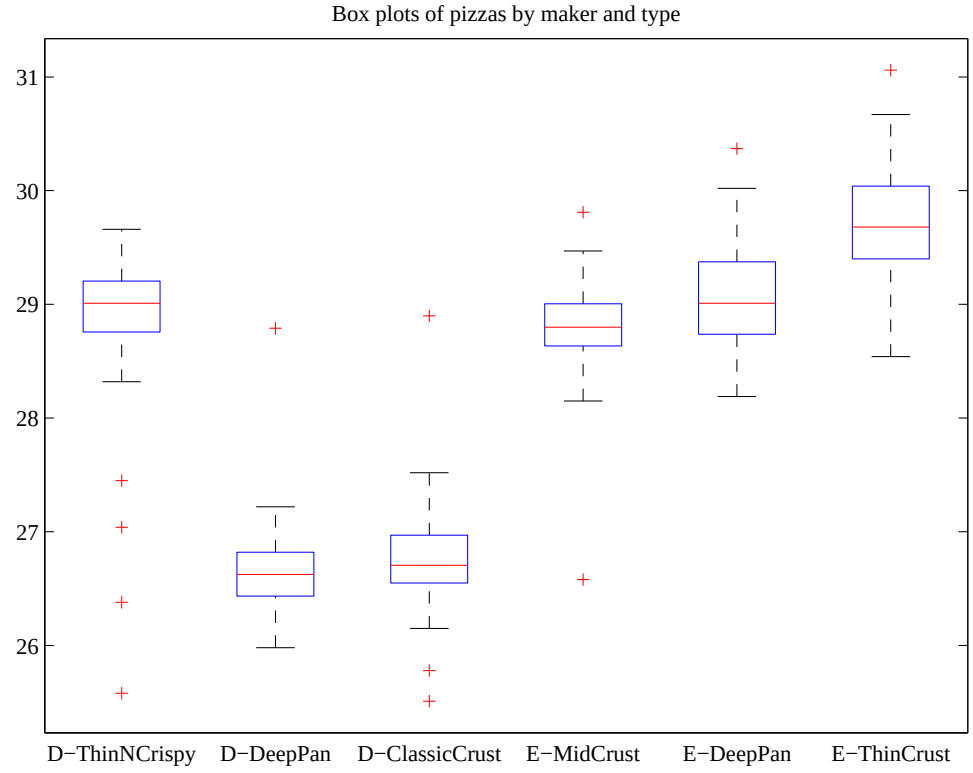


FIGURE 3.9: *Boxplots for the pizza data, broken out by type (thin crust, etc.).*

boxplot gives some more insight into the data. Dominos thin crust appear to have a narrow range of diameters (with several outliers), where the median pizza is rather larger than either the deep pan or the classic crust pizza. EagleBoys pizzas all have a range of diameters that is (a) rather similar across the types and (b) rather a lot like the Dominos thin crust. There are outliers, but few for each type.

Another possibility is that the variation in size is explained by the topping. We can compare types and toppings by producing a set of conditional boxplots (i.e. the diameters for each type and each topping). This leads to rather a lot of boxes (figure 3.10), but they're still easy to compare by eye. The main difficulty is that the labels on the plot have to be shortened. I made labels using the first letter from the manufacturer ("D" or "E"); the first letter from the crust type (previous paragraph); and the first and last letter of the topping. Toppings for Dominos are: Hawaiian; Supreme; BBQMeatlovers. For EagleBoys, toppings are: Hawaiian; SuperSupremo; and BBQMeatlovers. This gives the labels: 'DCBs'; (Dominos; ClassicCrust; BBQMeatlovers); 'DCHn'; 'DCSe'; 'DDBs'; 'DDHn'; 'DDSe'; 'DTBs'; 'DTHn'; 'DTSe'; 'EDBs'; 'EDHn'; 'EDSo'; 'EMBs'; 'EMHn'; 'EMSo'; 'ETBs'; 'ETHn'; 'ETSo'. Figure 3.10 suggests that the topping isn't what is important, but the crust (group the boxplots by eye).

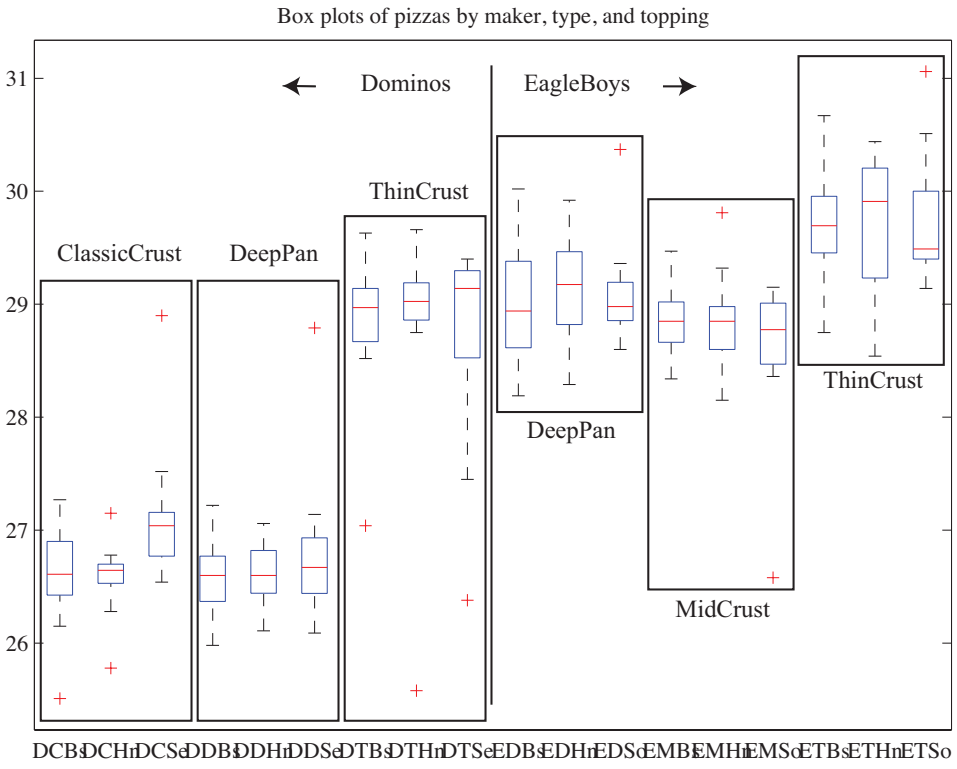


FIGURE 3.10: *The pizzas are now broken up by topping as well as crust type (look at the source for the meaning of the names). I have separated Dominos from Eagleboys with a vertical line, and grouped each crust type with a box. It looks as though the issue is not the type of topping, but the crust. Eagleboys seems to have tighter control over the size of the final pizza.*

What could be going on here? One possible explanation is that Eagleboys have tighter control over the size of the final pizza. One way this could happen is that all EagleBoys pizzas start the same size and shrink the same amount in baking, whereas all Dominos pizzas start a standard diameter, but different Dominos crusts shrink differently in baking. Another way is that Dominos makes different size crusts for different types, but that the cooks sometimes get confused. Yet another possibility is that Dominos controls portions by the mass of dough (so thin crust diameters tend to be larger), but Eagleboys controls by the diameter of the crust.

You should notice that this is more than just a fun story. If you were a manager at a pizza firm, you'd need to make choices about how to control costs. Labor costs, rent, and portion control (i.e. how much pizza, topping, etc. a customer gets for their money) are the main thing to worry about. If the same kind of pizza has a wide range of diameters, you have a problem, because some customers are getting too much (which affects your profit) or too little (which means they might call someone else). But making more regular pizzas might require more skilled (and so

more expensive) labor. The fact that Dominos and EagleBoys seem to be following different strategies successfully suggests that more than one strategy might work. But you can't choose if you don't know what's happening. As I said at the start, "what's going on here?" is perhaps the single most useful question anyone can ask.

## 3.4 NORMALIZED 2D SCATTER PLOTS

As you recall from section 2.3.4, scale is a problem for scatter plots. The way to avoid the problem is to plot in standard coordinates.

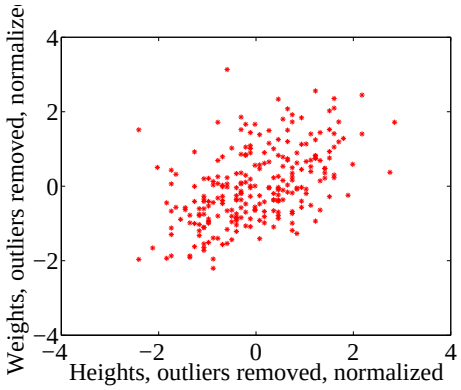


FIGURE 3.11: *A normalized scatter plot of weight against height, from the dataset at* `http://www2.stetson.edu/~jrasp/data.htm`. *Now you can see that someone who is a standard deviation taller than the mean will tend to be somewhat heavier than the mean too.*

A natural solution to problems with scale is to normalize the $x$ and $y$ coordinates of the two-dimensional data to standard coordinates. We can normalize without worrying about the dimension of the data — we normalize each dimension independently by subtracting the mean of that dimension and dividing by the standard deviation of that dimension. We continue to use the convention of writing the normalized $x$ coordinate as $\hat{x}$ and the normalized $y$ coordinate as $\hat{y}$. So, for example, we can write $\hat{x}_j = (x_j - \mathsf{mean}(\{x\}))/\mathsf{std}(x))$ for the $\hat{x}$ value of the $j$'th data item in normalized coordinates. Normalizing shows us the dataset on a standard scale. Once we have done this, it is quite straightforward to read off simple relationships between variables from a scatter plot.

## 3.5 CORRELATION

The simplest, and most important, relationship to look for in a scatter plot is this: when $\hat{x}$ increases, does $\hat{y}$ tend to increase, decrease, or stay the same? This is straightforward to spot in a normalized scatter plot, because each case produces a very clear shape on the scatter plot. Any relationship is called **correlation** (we will see later how to measure this), and the three cases are: positive correlation, which means that larger $\hat{x}$ values tend to appear with larger $\hat{y}$ values; zero correlation, which means no relationship; and negative correlation, which means that larger $\hat{x}$

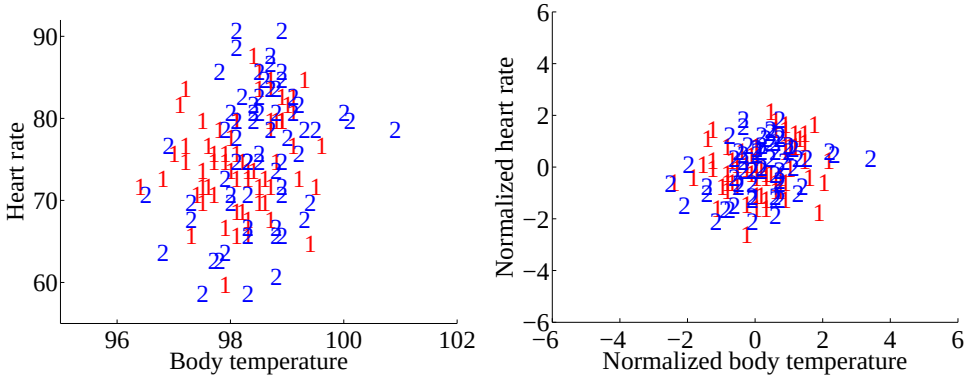


FIGURE 3.12: **Left:** *A scatter plot of body temperature against heart rate, from the dataset at* `http://www2.stetson.edu/~jrasp/data.htm`*; normtemp.xls. I have separated the two genders by plotting a different symbol for each (though I don't know which gender is indicated by which letter); if you view this in color, the differences in color makes for a greater separation of the scatter. This picture suggests, but doesn't conclusively establish, that there isn't much dependence between temperature and heart rate, and any dependence between temperature and heart rate isn't affected by gender. The scatter plot of the normalized data, in standard coordinates, on the* **right** *supports this view.*

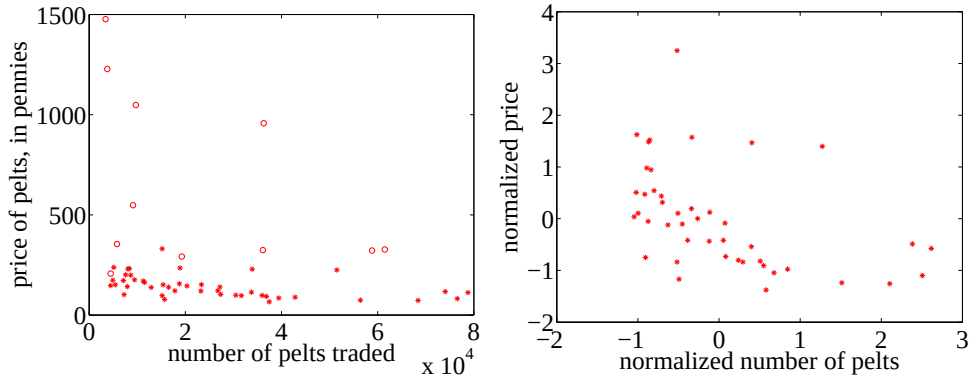


FIGURE 3.13: **Left:** *A scatter plot of the price of lynx pelts against the number of pelts (this is a repeat of figure 2.13 for reference). I have plotted data for 1901 to the end of the series as circles, and the rest of the data as *'s. It is quite hard to draw any conclusion from this data, because the scale is confusing.* **Right:** *A scatter plot of the price of pelts against the number of pelts for lynx pelts. I excluded data for 1901 to the end of the series, and then normalized both price and number of pelts. Notice that there is now a distinct trend; when there are fewer pelts, they are more expensive, and when there are more, they are cheaper.*

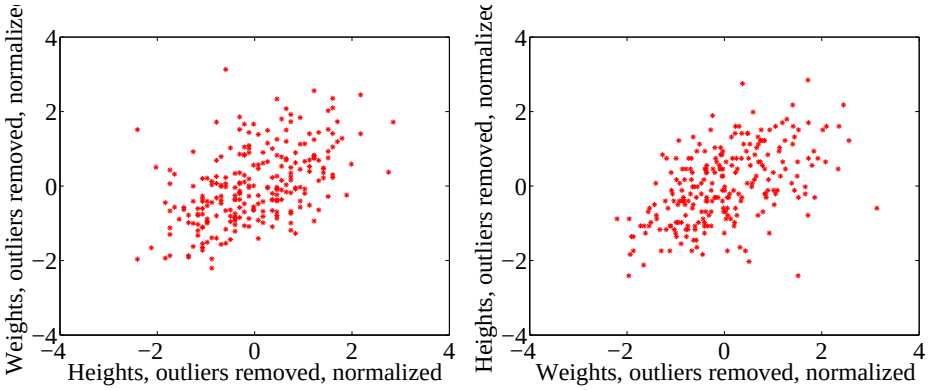


FIGURE 3.14: *On the* **left**, *a normalized scatter plot of weight (y-coordinate) against height (x-coordinate). On the* **right**, *a scatter plot of height (y-coordinate) against weight (x-coordinate). I've put these plots next to one another so you don't have to mentally rotate (which is what you should usually do).*

values tend to appear with smaller $\hat{y}$ values. I have shown these cases together in one figure using a real data example (Figure 3.15), so you can compare the appearance of the plots.

**Positive correlation** occurs when larger $\hat{x}$ values tend to appear with larger $\hat{y}$ values. This means that data points with with small (i.e. negative with large magnitude) $\hat{x}$ values must have small $\hat{y}$ values, otherwise the mean of $\hat{x}$ (resp. $\hat{y}$) would be too big. In turn, this means that the scatter plot should look like a "smear" of data from the bottom left of the graph to the top right. The smear might be broad or narrow, depending on some details we'll discuss below. Figure 3.11 shows normalized scatter plots of weight against height, and of body temperature against heart rate. In the weight-height plot, you can clearly see that individuals who are higher tend to weigh more. The important word here is "tend" — taller people could be lighter, but mostly they tend not to be. Notice, also, that I did NOT say that they weighed more *because* they were taller, but only that they tend to be heavier.

**Zero correlation** occurs when there is no relationship. This produces a characteristic shape in a scatter plot, but it takes a moment to understand why. If there really is no relationship, then knowing $\hat{x}$ will tell you nothing about $\hat{y}$. All we know is that $\mathsf{mean}\,(\{\hat{y}\}) = 0$, and $\mathsf{var}\,(\{\hat{y}\}) = 1$. Our value of $\hat{y}$ should have this mean and this variance, but it doesn't depend on $\hat{x}$ in any way. This is enough information to predict what the plot will look like. We know that $\mathsf{mean}\,(\{\hat{x}\}) = 0$ and $\mathsf{var}\,(\{\hat{x}\}) = 1$; so there will be many data points with $\hat{x}$ value close to zero, and few with a much larger or much smaller $\hat{x}$ value. The same applies to $\hat{y}$. Now consider the data points in a strip of $\hat{x}$ values. If this strip is far away from the origin, there will be few data points in the strip, because there aren't many big $\hat{x}$ values. If there is no relationship, we don't expect to see large or small $\hat{y}$ values in this strip, because there are few data points in the strip and because large or small $\hat{y}$ values are uncommon — we see them only if there are many data points,

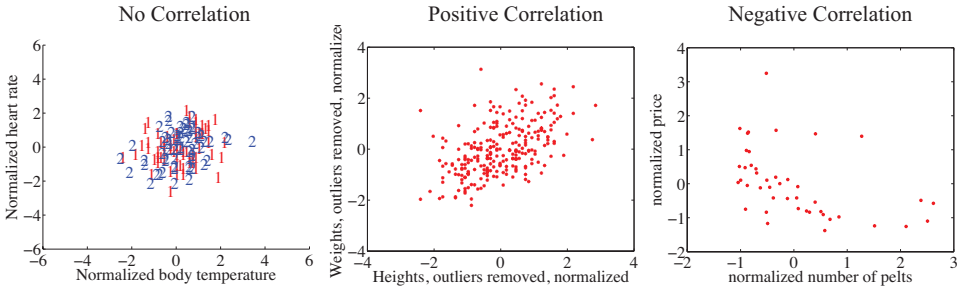


FIGURE 3.15: *The three kinds of scatter plot are less clean for real data than for our idealized examples. Here I used the body temperature vs heart rate data for the zero correlation; the height-weight data for positive correlation; and the lynx data for negative correlation. The pictures aren't idealized — real data tends to be messy — but you can still see the basic structures.*

and then seldom. So for a strip with $\hat{x}$ close to zero, we might see large $\hat{y}$ values; but for one that is far away, we expect to see small $\hat{y}$ values. We should see a blob, centered at the origin. In the temperature-heart rate plot of figure 3.12, it looks as though nothing of much significance is happening. The average heart rate seems to be about the same for people who run warm or who run cool. There is probably not much relationship here.

**Negative correlation** occurs when larger $\hat{x}$ values tend to appear with smaller $\hat{y}$ values. This means that data points with with small $\hat{x}$ values must have large $\hat{y}$ values, otherwise the mean of $\hat{x}$ (resp. $\hat{y}$) would be too big. In turn, this means that the scatter plot should look like a "smear" of data from the top left of the graph to the bottom right. The smear might be broad or narrow, depending on some details we'll discuss below. Figure 3.13 shows a normalized scatter plot of the lynx pelt-price data, where I have excluded the data from 1901 on. I did so because there seemed to be some other effect operating to drive prices up, which was inconsistent with the rest of the series. This plot suggests that when there were more pelts, prices were lower, as one would expect.

Notice that leaving out data, as I did here, should be done with care. If you exclude every data point that might disagree with your hypothesis, you may miss the fact that you are wrong. Leaving out data is an essential component of many kinds of fraud. You should always reveal whether you have excluded data, and why, to allow the reader to judge the evidence.

The correlation is not affected by which variable is plotted on the $x$-axis and which is plotted on the $y$-axis. Figure 3.14 compares a plot of height against weight to one of weight against height. Usually, one just does this by rotating the page, or by imagining the new picture. The left plot tells you that data points with higher height value tend to have higher weight value; the right plot tells you that data points with higher weight value tend to have higher height value — i.e. the plots tell you the same thing. It doesn't really matter which one you look at. Again, the important word is "tend" — the plot doesn't tell you anything about *why*, it just tells you that when one variable is larger the other tends to be, too.

### 3.5.1 The Correlation Coefficient

Consider a normalized data set of $N$ two-dimensional vectors. We can write the $i$'th data point *in standard coordinates* $(\hat{x}_i, \hat{y}_i)$. We already know many important summaries of this data, because it is in standard coordinates. We have $\mathsf{mean}\left(\{\hat{x}\}\right) = 0$; $\mathsf{mean}\left(\{\hat{y}\}\right) = 0$; $\mathsf{std}\left(\hat{x}\right) = 1$; and $\mathsf{std}\left(\hat{y}\right) = 1$. Each of these summaries is itself the mean of some monomial. So $\mathsf{std}\left(\hat{x}\right)^2 = \mathsf{mean}\left(\left\{\hat{x}^2\right\}\right) = 1$; $\mathsf{std}\left(\hat{y}\right)^2 = \mathsf{mean}\left(\left\{\hat{y}^2\right\}\right)$ (the other two are easy). We can rewrite this information in terms of means of monomials, giving $\mathsf{mean}\left(\{\hat{x}\}\right) = 0$; $\mathsf{mean}\left(\{\hat{y}\}\right) = 0$; $\mathsf{mean}\left(\left\{\hat{x}^2\right\}\right) = 1$; and $\mathsf{mean}\left(\left\{\hat{y}^2\right\}\right) = 1$. There is one monomial missing here, which is $\hat{x}\hat{y}$.

The term $\mathsf{mean}\left(\{\hat{x}\hat{y}\}\right)$ captures correlation between $x$ and $y$. The term is known as the **correlation coefficient** or **correlation**.

**Definition: 3.11** *Correlation coefficient*

Assume we have $N$ data items which are 2-vectors $(x_1, y_1), \ldots, (x_N, y_N)$, where $N > 1$. These could be obtained, for example, by extracting components from larger vectors. We compute the correlation coefficient by first normalizing the $x$ and $y$ coordinates to obtain $\hat{x}_i = \frac{(x_i - \mathsf{mean}(\{x\}))}{\mathsf{std}(x)}$, $\hat{y}_i = \frac{(y_i - \mathsf{mean}(\{y\}))}{\mathsf{std}(y)}$. The correlation coefficient is the mean value of $\hat{x}\hat{y}$, and can be computed as:

$$\mathsf{corr}\left(\{(x, y)\}\right) = \frac{\sum_i \hat{x}_i \hat{y}_i}{N}$$

Correlation is a measure of our ability to predict one value from another. The correlation coefficient takes values between $-1$ and 1 (we'll prove this below). If the correlation coefficient is close to 1, then we are likely to predict very well. Small correlation coefficients (under about 0.5, say, but this rather depends on what you are trying to achieve) tend not to be all that interesting, because (as we shall see) they result in rather poor predictions. Figure 3.16 gives a set of scatter plots of different real data sets with different correlation coefficients. These all come from data set of age-height-weight, which you can find at `http://www2.stetson.edu/~jrasp/data.htm` (look for bodyfat.xls). In each case, two outliers have been removed. Age and height are hardly correlated, as you can see from the figure. Younger people do tend to be slightly taller, and so the correlation coefficient is -0.25. You should interpret this as a small correlation. However, the variable called "adiposity" (which isn't defined, but is presumably some measure of the amount of fatty tissue) is quite strongly correlated with weight, with a correlation coefficient is 0.86. Average tissue density is quite strongly negatively correlated with adiposity, because muscle is much denser than fat, so these variables are negatively correlated — we expect high density to appear with low adiposity, and vice versa. The correlation coefficient is -0.86. Finally, density is very strongly correlated with body weight. The correlation coefficient is -0.98.

It's not always convenient or a good idea to produce scatter plots in standard

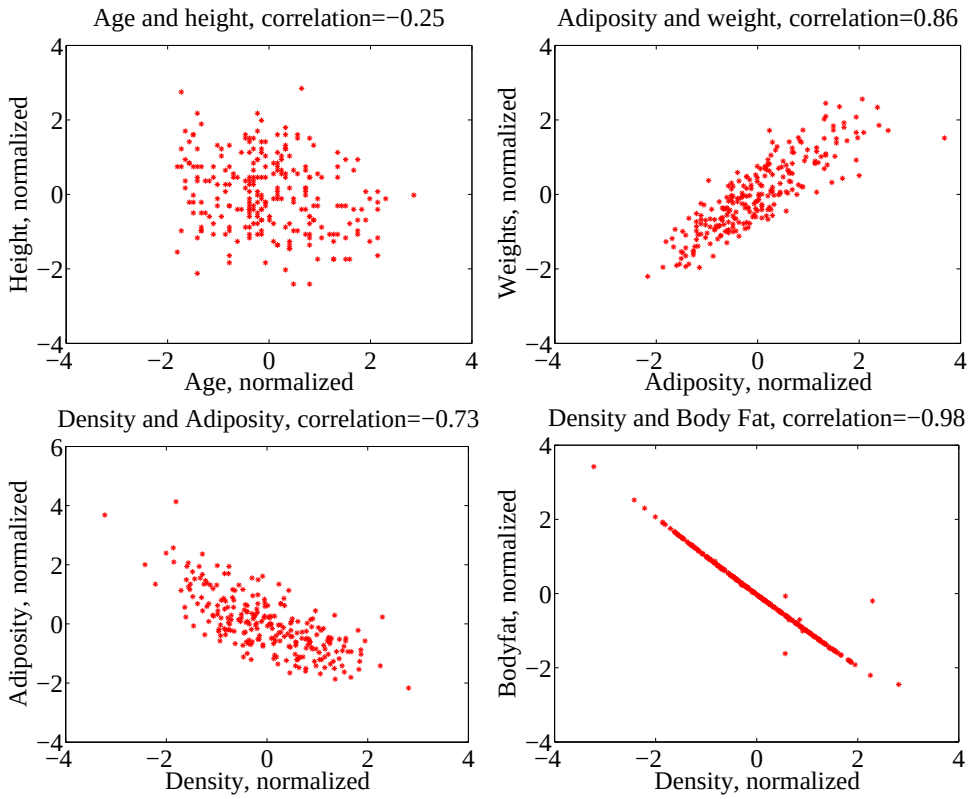


FIGURE 3.16: *Scatter plots for various pairs of variables for the age-height-weight dataset from* `http://www2.stetson.edu/~jrasp/data.htm`; *bodyfat.xls. In each case, two outliers have been removed, and the plots are in standard coordinates (compare to figure 3.17, which shows these data sets plotted in their original units). The legend names the variables.*

coordinates (among other things, doing so hides the units of the data, which can be a nuisance). Fortunately, scaling or translating data does not change the value of the correlation coefficient (though it can change the sign if one scale is negative). This means that it's worth being able to spot correlation in a scatter plot that isn't in standard coordinates (even though correlation is always *defined* in standard coordinates). Figure 3.17 shows different correlated datasets plotted in their original units. These data sets are the same as those used in figure 3.16

**Properties of the Correlation Coefficient**

You should memorize the following properties of the correlation coefficient:

- The correlation coefficient is symmetric (it doesn't depend on the order of its arguments), so

$$\mathsf{corr}\left(\{(x,y)\}\right) = \mathsf{corr}\left(\{(y,x)\}\right)$$

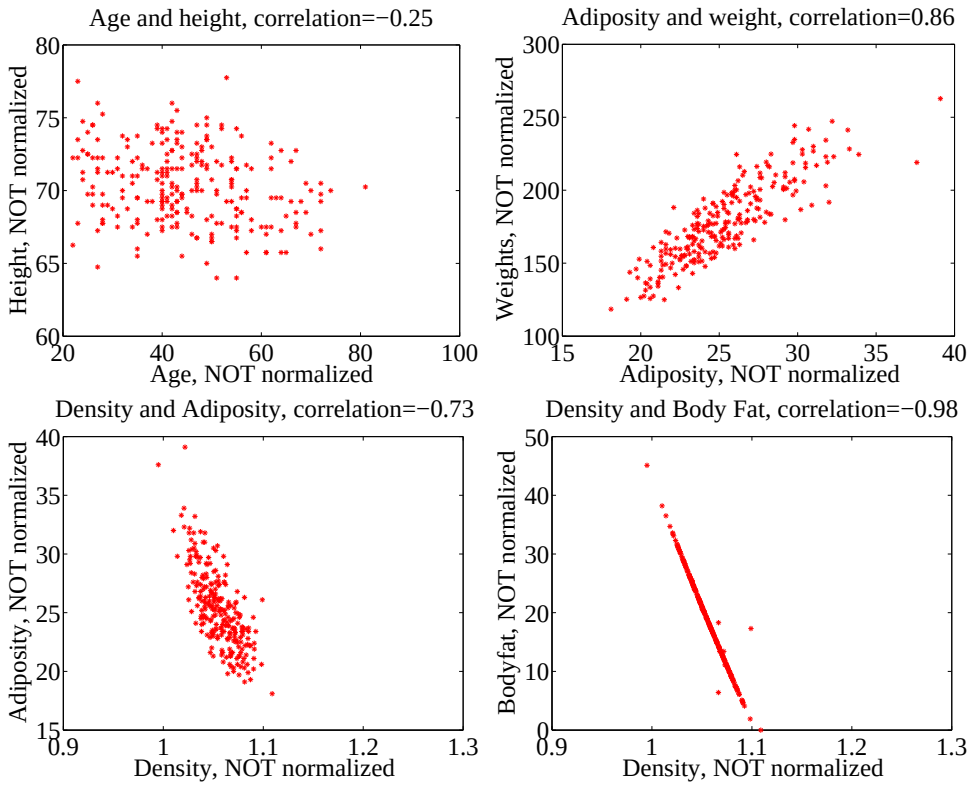


FIGURE 3.17: *Scatter plots for various pairs of variables for the age-height-weight dataset from* `http://www2.stetson.edu/~jrasp/data.htm`; *bodyfat.xls. In each case, two outliers have been removed, and the plots are* NOT *in standard coordinates (compare to figure 3.16, which shows these data sets plotted in normalized coordinates). The legend names the variables.*

- The value of the correlation coefficient is not changed by translating the data. Scaling the data can change the sign, but not the absolute value. For constants $a \neq 0$, $b$, $c \neq 0$, $d$ we have

$$\mathsf{corr}\left(\{(ax+b, cx+d)\}\right) = \text{sign}(ab)\mathsf{corr}\left(\{(x,y)\}\right)$$

- If $\hat{y}$ tends to be large (resp. small) for large (resp. small) values of $\hat{x}$, then the correlation coefficient will be positive.
- If $\hat{y}$ tends to be small (resp. large) for large (resp. small) values of $\hat{x}$, then the correlation coefficient will be negative.
- If $\hat{y}$ doesn't depend on $\hat{x}$, then the correlation coefficient is zero (or close to zero).
- The largest possible value is 1, which happens when $\hat{x} = \hat{y}$.

- The smallest possible value is -1, which happens when $\hat{x} = -\hat{y}$.

The first property is easy, and we relegate that to the exercises. One way to see that the correlation coefficient isn't changed by translation or scale is to notice that it is defined in standard coordinates, and scaling or translating data doesn't change those. Another way to see this is to scale and translate data, then write out the equations; notice that taking standard coordinates removes the effects of the scale and translation. In each case, notice that if the scale is negative, the sign of the correlation coefficient changes.

The property that, if $\hat{y}$ tends to be large (resp. small) for large (resp. small) values of $\hat{x}$, then the correlation coefficient will be positive, doesn't really admit a formal statement. But it's relatively straightforward to see what's going on. Because $\mathsf{mean}\,(\{\hat{x}\}) = 0$, small values of $\mathsf{mean}\,(\{\hat{x}\})$ must be negative and large values must be positive. But $\mathsf{corr}\,(\{(x,y)\}) = \frac{\sum_i \hat{x}_i \hat{y}_i}{N}$; and for this sum to be positive, it should contain mostly positive terms. It can contain few or no hugely positive (or hugely negative) terms, because $\mathsf{std}\,(\hat{x}) = \mathsf{std}\,(\hat{y}) = 1$ so there aren't many large (or small) numbers. For the sum to contain mostly positive terms, then the sign of $\hat{x}_i$ should be the same as the sign $\hat{y}_i$ for most data items. Small changes to this argument work to show that if if $\hat{y}$ tends to be small (resp. large) for large (resp. small) values of $\hat{x}$, then the correlation coefficient will be negative.

Showing that no relationship means zero correlation requires slightly more work. Divide the scatter plot of the dataset up into thin vertical strips. There are $S$ strips. Each strip is narrow, so the $\hat{x}$ value does not change much for the data points in a particular strip. For the $s$'th strip, write $N(s)$ for the number of data points in the strip, $\hat{x}(s)$ for the $\hat{x}$ value at the center of the strip, and $\overline{\hat{y}}(s)$ for the mean of the $\hat{y}$ values within that strip. Now the strips are narrow, so we can approximate all data points within a strip as having the same value of $\hat{x}$. This yields

$$\mathsf{mean}\,(\{\hat{x}\hat{y}\}) \approx \frac{1}{S} \sum_{s \in \text{strips}} \hat{x}(s) \left[ N(s) \overline{\hat{y}}(s) \right]$$

(where you could replace $\approx$ with $=$ if the strips were narrow enough). Now assume that $\hat{y}(s)$ does not change from strip to strip, meaning that there is no relationship between $\hat{x}$ and $\hat{y}$ in this dataset (so the picture is like the left hand side in figure 3.15). Then each value of $\overline{\hat{y}}(s)$ is the same — we write $\overline{\hat{y}}$ — and we can rearrange to get

$$\mathsf{mean}\,(\{\hat{x}\hat{y}\}) \approx \overline{\hat{y}} \frac{1}{S} \sum_{s \in \text{strips}} \hat{x}(s).$$

Now notice that

$$0 = \mathsf{mean}\,(\{\hat{y}\}) \approx \frac{1}{S} \sum_{s \in \text{strips}} N(s) \overline{\hat{y}}(s)$$

(where again you could replace $\approx$ with $=$ if the strips were narrow enough). This means that if every strip has the same value of $\overline{\hat{y}}(s)$, then that value must be zero. In turn, if there is no relationship between $\hat{x}$ and $\hat{y}$, we must have $\mathsf{mean}\,(\{\hat{x}\hat{y}\}) = 0$.

**Proposition:**

$$-1 \leq \mathsf{corr}\left(\{(x,y)\}\right) \leq 1$$

**Proof:** Writing $\hat{x}$, $\hat{y}$ for the normalized coefficients, we have

$$\mathsf{corr}\left(\{(x,y)\}\right) = \frac{\sum_i \hat{x}_i \hat{y}_i}{N}$$

and you can think of the value as the inner product of two vectors. We write

$$\mathbf{x} = \frac{1}{\sqrt{N}}\left[\hat{x}_1, \hat{x}_2, \ldots \hat{x}_N\right] \text{ and } \mathbf{y} = \frac{1}{\sqrt{N}}\left[\hat{y}_1, \hat{y}_2, \ldots \hat{y}_N\right]$$

and we have $\mathsf{corr}\left(\{(x,y)\}\right) = \mathbf{x}^T\mathbf{y}$. Notice $\mathbf{x}^T\mathbf{x} = \mathsf{std}\left(x\right)^2 = 1$, and similarly for $\mathbf{y}$. But the inner product of two vectors is at its maximum when the two vectors are the same, and this maximum is 1. This argument is also sufficient to show that smallest possible value of the correlation is $-1$, and this occurs when $\hat{x}_i = -\hat{y}_i$ for all $i$.

**Property 3.4:** The largest possible value of the correlation is 1, and this occurs when $\hat{x}_i = \hat{y}_i$ for all $i$. The smallest possible value of the correlation is $-1$, and this occurs when $\hat{x}_i = -\hat{y}_i$ for all $i$.

### 3.5.2 Using Correlation to Predict

Assume we have $N$ data items which are 2-vectors $(x_1, y_1), \ldots, (x_N, y_N)$, where $N > 1$. These could be obtained, for example, by extracting components from larger vectors. As usual, we will write $\hat{x}_i$ for $x_i$ in normalized coordinates, and so on. Now assume that we know the correlation coefficient is $r$ (this is an important, traditional notation). What does this mean?

One (very useful) interpretation is in terms of prediction. Assume we have a data point $(x_0, ?)$ where we know the $x$-coordinate, but not the $y$-coordinate. We can use the correlation coefficient to predict the $y$-coordinate. First, we transform to standard coordinates. Now we must obtain the best $\hat{y}_0$ value to predict, using the $\hat{x}_0$ value we have.

We want to construct a prediction function which gives a prediction for any value of $\hat{x}$. This predictor should behave as well as possible on our existing data. For each of the $(\hat{x}_i, \hat{y}_i)$ pairs in our data set, the predictor should take $\hat{x}_i$ and produce a result as close to $\hat{y}_i$ as possible. We can choose the predictor by looking at the errors it makes at each data point.

We write $\hat{y}_i^p$ for the value of $\hat{y}_i$ predicted at $\hat{x}_i$. The simplest form of predictor is linear. If we predict using a linear function, then we have, for some unknown $a$, $b$, that $\hat{y}_i^p = a\hat{x}_i + b$. Now think about $u_i = \hat{y}_i - \hat{y}_i^p$, which is the error in our prediction. We would like to have $\mathsf{mean}\left(\{u\}\right) = 0$ (otherwise, we could reduce the

error of the prediction just by subtracting a constant).

$$\begin{aligned}
\mathsf{mean}\left(\{u\}\right) &= \mathsf{mean}\left(\{\hat{y} - \hat{y}^p\}\right) \\
&= \mathsf{mean}\left(\{\hat{y}\}\right) - \mathsf{mean}\left(\{a\hat{x}_i + b\}\right) \\
&= \mathsf{mean}\left(\{\hat{y}\}\right) - a\mathsf{mean}\left(\{\hat{x}\}\right) + b \\
&= 0 - a0 + b \\
&= 0.
\end{aligned}$$

This means that we must have $b = 0$.

To estimate $a$, we need to think about $\mathsf{var}\left(\{u\}\right)$. We should like $\mathsf{var}\left(\{u\}\right)$ to be as small as possible, so that the errors are as close to zero as possible (remember, small variance means small standard deviation which means the data is close to the mean). We have

$$\begin{aligned}
\mathsf{var}\left(\{u\}\right) &= \mathsf{var}\left(\{\hat{y} - \hat{y}^p\}\right) \\
&= \mathsf{mean}\left(\left\{(\hat{y} - a\hat{x})^2\right\}\right) \quad \text{because } \mathsf{mean}\left(\{u\}\right) = 0 \\
&= \mathsf{mean}\left(\left\{(\hat{y})^2 - 2a\hat{x}\hat{y} + a^2(\hat{x})^2\right\}\right) \\
&= \mathsf{mean}\left(\left\{(\hat{y})^2\right\}\right) - 2a\mathsf{mean}\left(\{\hat{x}\hat{y}\}\right) + a^2\mathsf{mean}\left(\left\{(\hat{x})^2\right\}\right) \\
&= 1 - 2ar + a^2,
\end{aligned}$$

which we want to minimize by choice of $a$. At the minimum, we must have

$$\frac{d\mathsf{var}\left(\{u_i\}\right)}{da} = 0 = -2r + 2a$$

so that $a = r$ and the correct prediction is

$$\hat{y}_0^p = r\hat{x}_0$$

You can use a version of this argument to establish that if we have $(?, \hat{y}_0)$, then the best prediction for $\hat{x}_0$ (*which is in standard coordinates*) is $r\hat{y}_0$. It is important to notice that the coefficient of $\hat{y}_i$ is NOT $1/r$; you should work this example, which appears in the exercises. We now have a prediction procedure, outlined below.

**Procedure: 3.1** *Predicting a value using correlation*

Assume we have $N$ data items which are 2-vectors $(x_1, y_1), \ldots, (x_N, y_N)$, where $N > 1$. These could be obtained, for example, by extracting components from larger vectors. Assume we have an $x$ value $x_0$ for which we want to give the best prediction of a $y$ value, based on this data. The following procedure will produce a prediction:

- Transform the data set into standard coordinates, to get

$$\begin{aligned}\hat{x}_i &= \frac{1}{\mathsf{std}\,(x)}(x_i - \mathsf{mean}\,(\{x\})) \\ \hat{y}_i &= \frac{1}{\mathsf{std}\,(y)}(y_i - \mathsf{mean}\,(\{y\})) \\ \hat{x}_0 &= \frac{1}{\mathsf{std}\,(x)}(x_0 - \mathsf{mean}\,(\{x\})).\end{aligned}$$

- Compute the correlation

$$r = \mathsf{corr}\,(\{(x, y)\}) = \mathsf{mean}\,(\{\hat{x}\hat{y}\}).$$

- Predict $\hat{y}_0 = r\hat{x}_0$.
- Transform this prediction into the original coordinate system, to get

$$y_0 = \mathsf{std}\,(y) r \hat{x}_0 + \mathsf{mean}\,(\{y\})$$

Now assume we have a $y$ value $y_0$, for which we want to give the best prediction of an $x$ value, based on this data. The following procedure will produce a prediction:

- Transform the data set into standard coordinates.
- Compute the correlation.
- Predict $\hat{x}_0 = r\hat{y}_0$.
- Transform this prediction into the original coordinate system, to get

$$x_0 = \mathsf{std}\,(x) r \hat{y}_0 + \mathsf{mean}\,(\{x\})$$

There is another way of thinking about this prediction procedure, which is often helpful. Assume we need to predict a value for $x_0$. In normalized coordinates, our prediction is $\hat{y}^p = r\hat{x}_0$; if we revert back to the original coordinate system, the

prediction becomes

$$\frac{(y^p - \mathsf{mean}\,(\{y\}))}{\mathsf{std}\,(y)} = r(\frac{(x_0 - \mathsf{mean}\,(\{x\}))}{\mathsf{std}\,(x)}).$$

This gives a really useful rule of thumb, which I have broken out in the box below.

**Procedure: 3.2** *Predicting a value using correlation: Rule of thumb - 1*

If $x_0$ is $k$ standard deviations from the mean of $x$, then the predicted value of $y$ will be $rk$ standard deviations away from the mean of $y$, and the sign of $r$ tells whether $y$ increases or decreases.

An even more compact version of the rule of thumb is in the following box.

**Procedure: 3.3** *Predicting a value using correlation: Rule of thumb - 2*

The predicted value of $y$ goes up by $r$ standard deviations when the value of $x$ goes up by one standard deviation.

We can compute the average root mean square error that this prediction procedure will make. The square of this error must be

$$\begin{aligned}\mathsf{mean}\left(\left\{u^2\right\}\right) &= \mathsf{mean}\left(\left\{y^2\right\}\right) - 2r\mathsf{mean}\left(\{xy\}\right) + r^2\mathsf{mean}\left(\left\{x^2\right\}\right) \\ &= 1 - 2r^2 + r^2 \\ &= 1 - r^2\end{aligned}$$

so the root mean square error will be $\sqrt{1-r^2}$. This is yet another intepretation of correlation; if $x$ and $y$ have correlation close to one, then predictions could have very small root mean square error, and so might be very accurate. In this case, knowing one variable is about as good as knowing the other. If they have correlation close to zero, then the root mean square error in a prediction might be as large as the root mean square error in $\hat{y}$ — which means the prediction is nearly a pure guess.

The prediction argument means that we can spot correlations for data in other kinds of plots — one doesn't have to make a scatter plot. For example, if we were to observe a child's height from birth to their 10'th year (you can often find these observations in ballpen strokes, on kitchen walls), we could plot height as a function of year. If we also had their weight (less easily found), we could plot weight as a function of year, too. The prediction argument above say that, if you can predict the weight from the height (or vice versa) then they're correlated. One way to spot this is to look and see if one curve goes up when the other does (or goes down when the other goes up). You can see this effect in figure 2.7, where (before 19h00), prices go down when the number of pelts goes up, and vice versa. These two variables are negatively correlated.

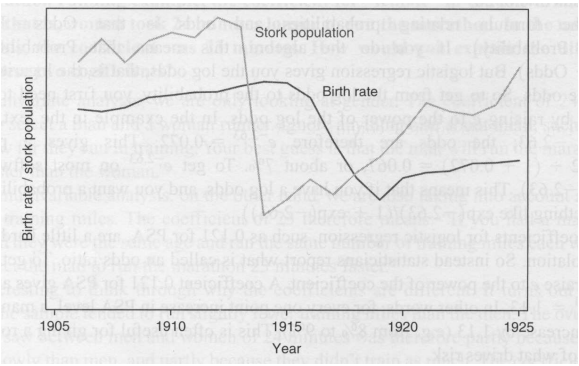


FIGURE 3.18: *This figure, from Vickers (ibid, p184) shows a plot of the stork population as a function of time, and the human birth rate as a function of time, for some years in Germany. The correlation is fairly clear; but this does not mean that reducing the number of storks means there are fewer able to bring babies. Instead, this is the impact of the first world war — a hidden or latent variable.*

### 3.5.3 Confusion caused by correlation

There is one very rich source of potential (often hilarious) mistakes in correlation. When two variables are correlated, they change together. If the correlation is positive, that means that, in typical data, if one is large then the other is large, and if one is small the other is small. In turn, this means that one can make a reasonable prediction of one from the other. However, correlation DOES NOT mean that changing one variable causes the other to change (sometimes known as causation).

Two variables in a dataset could be correlated for a variety of reasons. One important reason is pure accident. If you look at enough pairs of variables, you may well find a pair that appears to be correlated just because you have a small set of observations. Imagine, for example, you have a dataset consisting of only two vectors — there is a pretty good chance that there is some correlation between the coefficients. Such accidents can occur in large datasets, particularly if the dimensions are high.

Another reason variables could be correlated is that there is some causal relationship — for example, pressing the accelerator tends to make the car go faster, and so there will be some correlation between accelerator position and car acceleration. As another example, adding fertilizer does tend to make a plant grow bigger. Imagine you record the amount of fertilizer you add to each pot, and the size of the resulting potplant. There should be some correlation.

Yet another reason variables could be correlated is that there is some other background variable — often called a **latent variable** — linked causally to each of the observed variables. For example, in children (as Freedman, Pisani and Purves note in their excellent *Statistics*), shoe size is correlated with reading skills. This DOES NOT mean that making your feet grow will make you read faster, or that you can make your feet shrink by forgetting how to read. The real issue here is

the age of the child. Young children tend to have small feet, and tend to have weaker reading skills (because they've had less practice). Older children tend to have larger feet, and tend to have stronger reading skills (because they've had more practice). You can make a reasonable prediction of reading skills from foot size, because they're correlated, even though there is no direct connection.

This kind of effect can mask correlations, too. Imagine you want to study the effect of fertilizer on potplants. You collect a set of pots, put one plant in each, and add different amounts of fertilizer. After some time, you record the size of each plant. You expect to see correlation between fertilizer amount and plant size. But you might not if you had used a different species of plant in each pot. Different species of plant can react quite differently to the same fertilizer (some plants just die if over-fertilized), so the species could act as a latent variable. With an unlucky choice of the different species, you might even conclude that there was a negative correlation between fertilizer and plant size. This example illustrates why you need to take great care in setting up experiments and interpreting their results.

This sort of thing happens often, and it's an effect you should look for. Another nice example comes from Vickers (ibid). The graph, shown in Figure 3.18, shows a plot of (a) a dataset of the stork population in Europe over a period of years and (b) a dataset of the birth rate over those years. This isn't a scatter plot; instead, the data has been plotted on a graph. You can see by eye that these two datasets are quite strongly correlated . Even more disturbing, the stork population dropped somewhat before the birth rate dropped. Is this evidence that storks brought babies in Europe during those years? No (the usual arrangement seems to have applied). For a more sensible explanation, look at the dates. The war disturbed both stork and human breeding arrangements. Storks were disturbed immediately by bombs, etc., and the human birth rate dropped because men died at the front.

## 3.6 STERILE MALES IN WILD HORSE HERDS

Large herds of wild horses are (apparently) a nuisance, but keeping down numbers by simply shooting surplus animals would provoke outrage. One strategy that has been adopted is to sterilize males in the herd; if a herd contains sufficient sterile males, fewer foals should result. But catching stallions, sterilizing them, and reinserting them into a herd is a performance — does this strategy work?

We can get some insight by plotting data. At `http://lib.stat.cmu.edu/DASL/Datafiles/WildHorses.html`, you can find a dataset covering herd management in wild horses. I have plotted part of this dataset in figure 3.19. In this dataset, there are counts of all horses, sterile males, and foals made on each of a small number of days in 1986, 1987, and 1988 for each of two herds. I extracted data for one herd. I have plotted this data as a function of the count of days since the first data point, because this makes it clear that some measurements were taken at about the same time, but there are big gaps in the measurements. In this plot, the data points are shown with a marker. Joining them leads to a confusing plot because the data points vary quite strongly. However, notice that the size of the herd drifts down slowly (you could hold a ruler against the plot to see the trend), as does the number of foals, when there is a (roughly) constant number of sterile

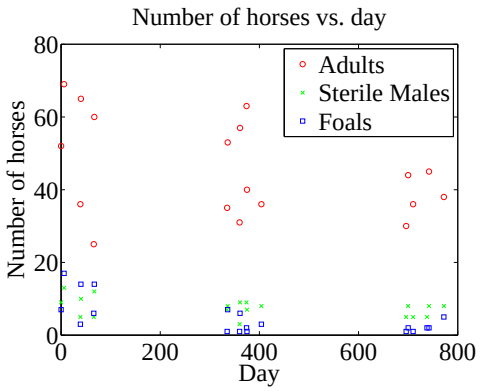


FIGURE 3.19: *A plot of the number of adult horses, sterile males, and foals in horse herds over a period of three years. The plot suggests that introducing sterile males might cause the number of foals to go down. Data from* `http://lib.stat.cmu.edu/DASL/Datafiles/WildHorses.html`.

males.

Does sterilizing males result in fewer foals? This is likely hard to answer for this dataset, but we could ask whether herds with more sterile males have fewer foals. A scatter plot is a natural tool to attack this question. However, the scatter plots of figure 3.20 suggest, rather surprisingly, that when there are more sterile males there are more adults (and vice versa), and when there are more sterile males there are more foals (and vice versa). This is borne out by a correlation analysis. The correlation coefficient between foals and sterile males is 0.74, and the correlation coefficient between adults and sterile males is 0.68. You should find this very surprising — how do the horses know how many sterile males there are in the herd? You might think that this is an effect of scaling the plot, but there is a scatter plot in normalized coordinates in figure 3.20 that is entirely consistent with the conclusions suggested by the unnormalized plot. What is going on here?

The answer is revealed by the scatter plots of figure 3.21. Here, rather than plotting a '*' at each data point, I have plotted the day number of the observation. This is in days from the first observation. You can see that the whole herd is shrinking — observations where there are many adults (resp. sterile adults, foals) occur with small day numbers, and observations where there are few have large day numbers. Because the whole herd is shrinking, it is true that when there are more adults and more sterile males, there are also more foals. Alternatively, you can see the plots of figure 3.19 as a scatter plot of herd size (resp. number of foals, number of sterile males) against day number. Then it becomes clear that the whole herd is shrinking, as is the size of each group. To drive this point home, we can look at the correlation coefficient between adults and days (-0.24), between sterile adults and days (-0.37), and between foals and days (-0.61). We can use the rule of thumb in box 3 to interpret this. This means that every 282 days, the herd loses about three adults; about one sterile adult; and about three foals. For the herd to have a stable size, it needs to gain by birth as many foals as it loses both to growing up

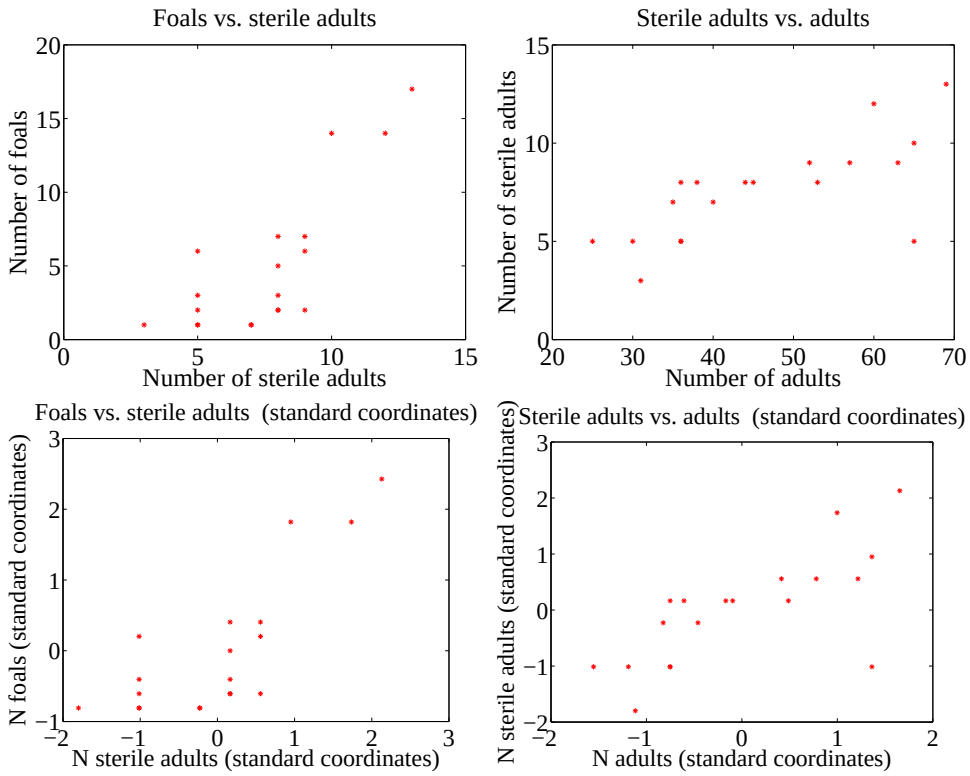


FIGURE 3.20: *Scatter plots of the number of sterile males in a horse herd against the number of adults, and the number of foals against the number of sterile males, from data of* `http://lib.stat.cmu.edu/DASL/Datafiles/WildHorses.html`. **Top:** *unnormalized;* **bottom:** *standard coordinates.*

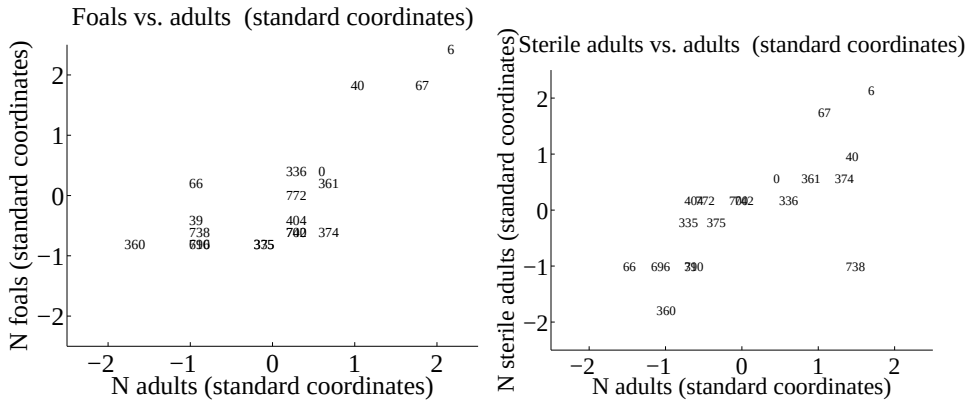


FIGURE 3.21:

and to death. If the herd is losing three foals every 282 days, then if they all grow up to replace the missing adults, the herd will be shrinking slightly (because it is losing four adults in this time); but if it loses foals to natural accidents, etc., then it is shrinking rather fast.

The message of this example is important. To understand a simple dataset, you might need to plot it several ways. You should make a plot, look at it and ask what it says, and then try to use another type of plot to confirm or refute what you think might be going on.

CHAPTER 4

# Visualizing High Dimensional Data

Chapter **??** described methods to explore the relationship between two elements in a dataset. We could extract a pair of elements and construct various plots. For vector data, we could also compute the correlation between different pairs of elements. But if each data item is $d$-dimensional, there could be a lot of pairs to deal with.

We will think of our dataset as a collection of $d$ dimensional vectors. It turns out that there are easy generalizations of our summaries. However, is hard to plot $d$-dimensional vectors. We need to find some way to make them fit on a 2-dimensional plot. Some simple methods can offer insights, but to really get what is going on we need methods that can at all pairs of relationships in a dataset in one go.

These methods visualize the dataset as a "blob" in a $d$-dimensional space. Many such blobs are flattened in some directions, because components of the data are strongly correlated. Finding the directions in which the blobs are flat yields methods to compute lower dimensional representations of the dataset.

## 4.1 SUMMARIES AND SIMPLE PLOTS

In this chapter, we assume that our data items are vectors. This means that we can add and subtract values and multiply values by a scalar without any distress. This is an important assumption, but it doesn't necessarily mean that data is continuous (for example, you can meaningfully add the number of children in one family to the number of children in another family). It does rule out a lot of discrete data. For example, you can't add "sports" to "grades" and expect a sensible answer.

**Notation:** Our data items are vectors, and we write a vector as $\mathbf{x}$. The data items are $d$-dimensional, and there are $N$ of them. The entire data set is $\{\mathbf{x}\}$. When we need to refer to the $i$'th data item, we write $\mathbf{x}_i$. We write $\{\mathbf{x}_i\}$ for a new dataset made up of $N$ items, where the $i$'th item is $\mathbf{x}_i$. If we need to refer to the $j$'th component of a vector $\mathbf{x}_i$, we will write $x_i^{(j)}$ (notice this isn't in bold, because it is a component not a vector, and the $j$ is in parentheses because it isn't a power). Vectors are always column vectors.

### 4.1.1 The Mean

For one-dimensional data, we wrote

$$\mathsf{mean}\left(\{x\}\right) = \frac{\sum_i x_i}{N}.$$

This expression is meaningful for vectors, too, because we can add vectors and divide by scalars. We write

$$\text{mean}\left(\{\mathbf{x}\}\right) = \frac{\sum_i \mathbf{x}_i}{N}$$

and call this the mean of the data. Notice that each component of $\text{mean}\left(\{\mathbf{x}\}\right)$ is the mean of that component of the data. There is not an easy analogue of the median, however (how do you order high dimensional data?) and this is a nuisance. Notice that, just as for the one-dimensional mean, we have

$$\text{mean}\left(\{\mathbf{x} - \text{mean}\left(\{\mathbf{x}\}\right)\}\right) = 0$$

(i.e. if you subtract the mean from a data set, the resulting data set has zero mean).

### 4.1.2 Parallel Plots

Parallel plots can sometimes reveal information, particularly when the dimension of the dataset is low. To construct a parallel plot, you compute a normalized representation of each component of each data item. The component is normalized by translating and scaling so that the minimum value over the dataset is zero, and the maximum value over the dataset is one. Now write the $i$'th normalised data item as $(n_1, n_2, \ldots, n_d)$. For this data item, you plot a broken line joining $(1, n_1)$ to $(2, n_2)$ to $(3, n_3$, etc. These plots are superimposed on one another. In the case of the bodyfat dataset, this yields the plot of figure 4.1.

Some structures in the parallel plot are revealing. Outliers often stick out (in figure 4.1, it's pretty clear that there's a data point with a very low height value, and also one with a very large weight value). Outliers affect the scaling, and so make other structures difficult to spot. I have removed them for figure 4.2. In this figure, you can see that two negatively correlated components next to one another produce a butterfly like shape (bodyfat and density). In this plot, you can also see that there are still several data points that are very different from others (two data items have ankle values that are very different from the others, for example).

### 4.1.3 Understanding Blobs with Scatterplot Matrices

Plotting high dimensional data is tricky. One strategy that is very useful when there aren't too many dimensions is to use a scatterplot matrix. To build one, you lay out scatterplots for each pair of variables in a matrix. On the diagonal, you name the variable that is the vertical axis for each plot in the row, and the horizontal axis in the column. This sounds more complicated than it is; look at the example of figure 4.3, which shows a scatterplot matrix for four of the variables in the height weight dataset of `http://www2.stetson.edu/~jrasp/data.htm`; look for bodyfat.xls at that URL). This is originally a 16-dimensional dataset, but a 16 by 16 scatterplot matrix is squashed and hard to interpret.

What is nice about this kind of plot is that it's quite easy to spot correlations between pairs of variables, though you do need to take into account the coordinates have not been normalized. For figure 4.3, you can see that weight and adiposity appear to show quite strong correlations, but weight and age are pretty weakly

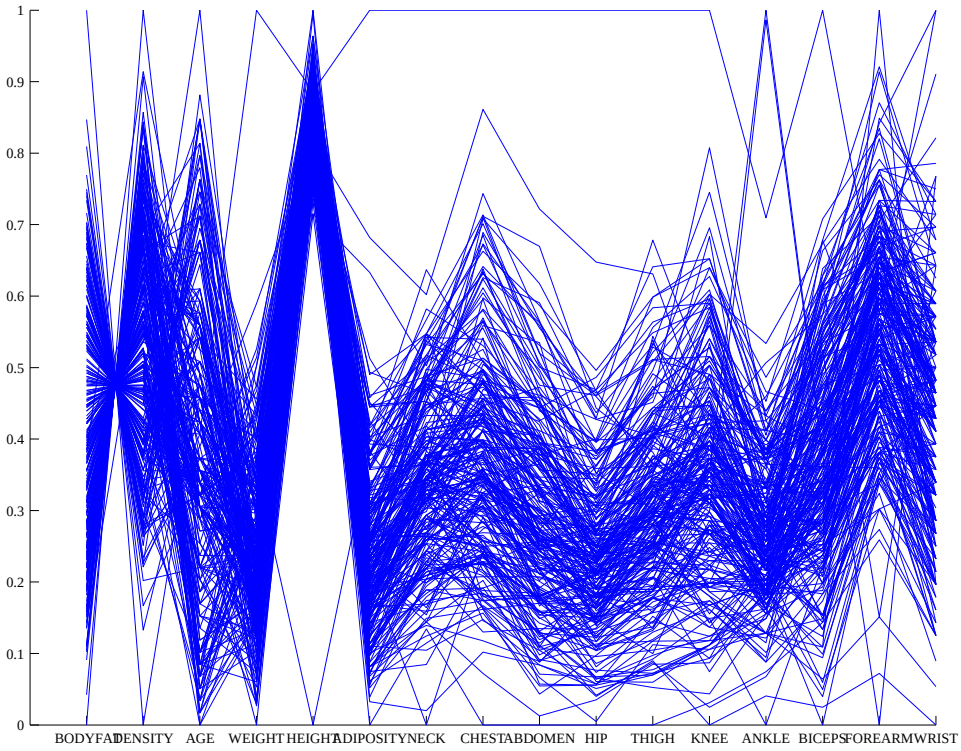


FIGURE 4.1: *A parallel plot of the bodyfat dataset, including all data points. I have named the components on the horizontal axis. It is easy to see that large values of bodyfat correspond to small values of density, and vice versa. Notice that one datapoint has height very different from all others; similarly, one datapoint has weight very different from all others.*

correlated. Height and age seem to have a low correlation. It is also easy to visualize unusual data points. Usually one has an interactive process to do so — you can move a "brush" over the plot to change the color of data points under the brush. To show what might happen, figure 4.4 shows a scatter plot matrix with some points shown as circles. Notice how they lie inside the "blob" of data in some views, and outside in others. This is an effect of projection.

UC Irvine keeps a large repository of datasets that are important in machine learning. You can find the repository at `http://archive.ics.uci.edu/ml/index.html`. Figures 4.5 and 4.6 show visualizations of a famous dataset to do with the botanical classification of irises.

Figures **??**, **??** and 4.13 show visualizations of another dataset to do with forest fires in Portugal, also from the UC Irvine repository (look at `http://archive.ics.uci.edu/ml/datasets/Forest+Fires`). In this dataset, there are a variety of measurements of location, time, temperature, etc. together with the area burned by a wildfire. It would be nice to know what leads to large fires, and a visualization

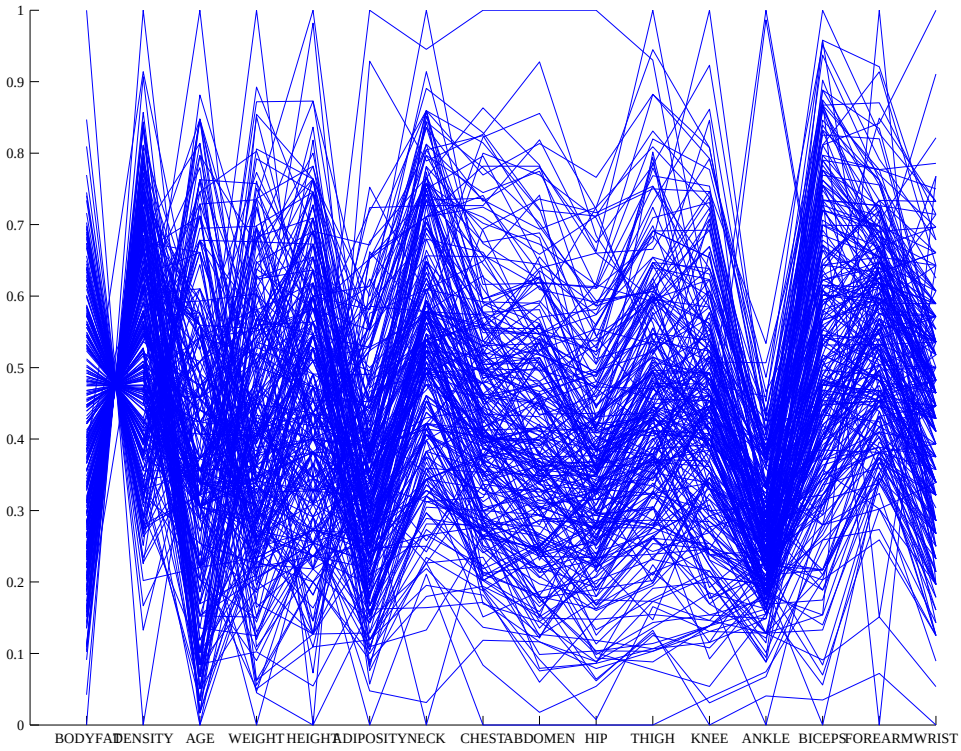


FIGURE 4.2: *A plot with those data items removed, so that those components are renormalized. Two datapoints have rather distinct ankle measurements. Generally, you can see that large knees go with large ankles and large biceps (the v structure).*

is the place to start. Many fires are tiny (or perhaps there was no area measurement?) and so many values of the area are zero. I found it helpful to take the log of area, and then to divide the values of the logarithm into seven categories. I ignored the first four variables, because I didn't think they'd be too important. **Exercise:** was I right? I made two scatterplot matrices, because an eight by eight matrix is too big to view. Generally, this visualization suggests that it would be hard to predict the size of a fire from these variables.

We can combine tools to analyze datasets. In the UC Irvine repository, you can find a dataset related to heart disease (look at `http://archive.ics.uci.edu/ml/datasets/Heart+Disease`). There are a variety of versions of the dataset; I used the version in the file "processed.cleveland.data". This contains a set of 14 features describing individuals under study. The 14'th is a measure of heart disease. What can we learn from this dataset?

The first thing to notice is that many of the variables are categorical. It is natural to make some mosaic plots to visualize what is happening. I quantized the age to five levels (0-20, 20-40, etc.), and quantized the measure of heart disease to two levels (no disease and disease) to simplify the plot. Figure **??** shows a mosaic

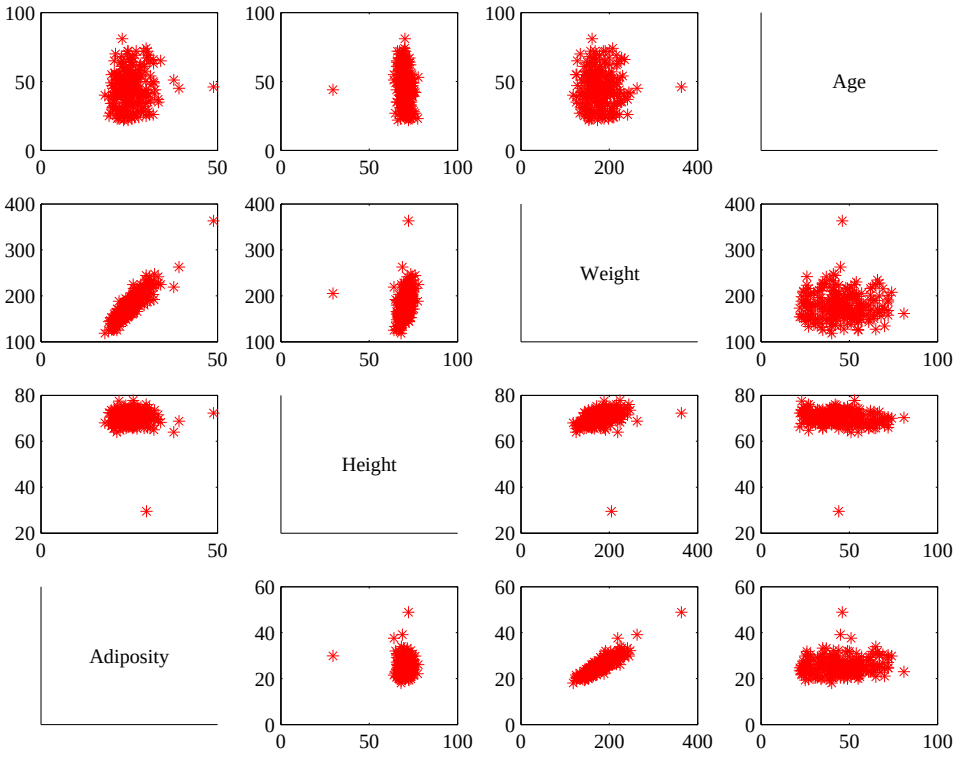


FIGURE 4.3: *This is a scatterplot matrix for four of the variables in the height weight dataset of* `http://www2.stetson.edu/~jrasp/data.htm`. *Each plot is a scatterplot of a pair of variables. The name of the variable for the horizontal axis is obtained by running your eye down the column; for the vertical axis, along the row. Although this plot is redundant (half of the plots are just flipped versions of the other half), that redundancy makes it easier to follow points by eye. You can look at a column, move down to a row, move across to a column, etc. Notice how you can spot correlations between variables and outliers (the arrows).*

plot of the result.

### 4.1.4 Using Covariance to encode Variance and Correlation

Variance, standard deviation and correlation can each be seen as an instance of a more general operation on data. Assume that we have two one dimensional data sets $\{x\}$ and $\{y\}$. Then we can define the **covariance** of $\{x\}$ and $\{y\}$.

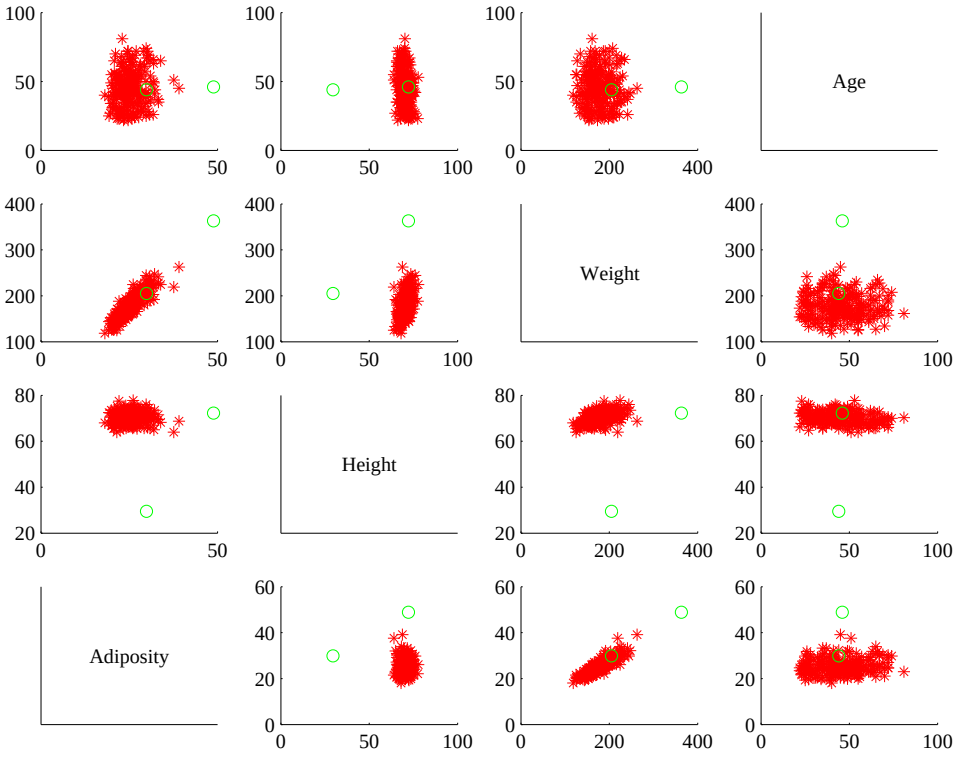


FIGURE 4.4: *You should compare this figure with figure 4.3. I have marked two data points with circles in this figure; notice that in some panels these are far from the rest of the data, in others close by. A "brush" in an interactive application can be used to mark data like this to allow a user to explore a dataset.*

**Definition: 4.1** *Covariance*

Assume we have two sets of $N$ data items, $\{x\}$ and $\{y\}$. We compute the covariance by

$$\mathsf{cov}\left(\{x\},\{y\}\right) = \frac{\sum_i (x_i - \mathsf{mean}\left(\{x\}\right))(y_i - \mathsf{mean}\left(\{y\}\right))}{N}$$

Covariance measures the tendency of corresponding elements of $\{x\}$ and of $\{y\}$ to be larger than (resp. smaller than) the mean. Just like mean, standard deviation and variance, covariance can refer either to a property of a dataset (as in the definition here) or a particular expectation (as in chapter **??**). The correspondence is defined by the order of elements in the data set, so that $x_1$ corresponds to $y_1$,

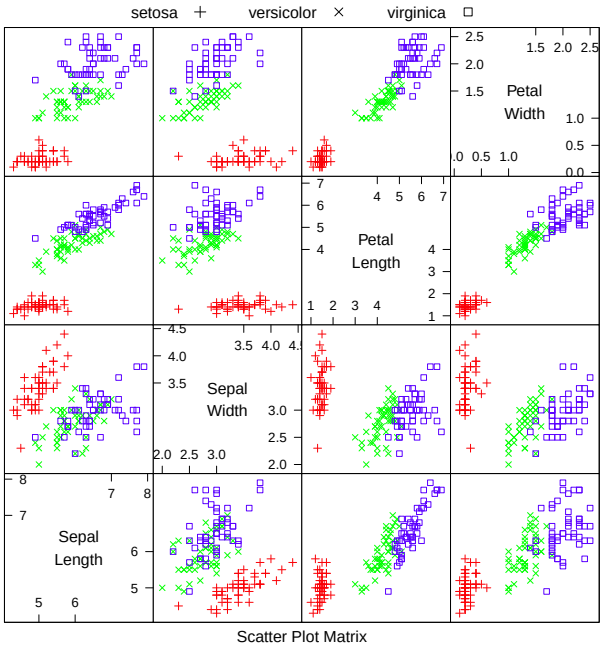


FIGURE 4.5: *This is a scatterplot matrix for the famous Iris data, originally due to ***. There are four variables, measured for each of three species of iris. I have plotted each species with a different marker. You can see from the plot that the species cluster quite tightly, and are different from one another. R code for this plot is on the website.*

$x_2$ corresponds to $y_2$, and so on. If $\{x\}$ tends to be larger (resp. smaller) than its mean for data points where $\{y\}$ is also larger (resp. smaller) than its mean, then the covariance should be positive. If $\{x\}$ tends to be larger (resp. smaller) than its mean for data points where $\{y\}$ is smaller (resp. larger) than its mean, then the covariance should be negative.

From this description, it should be clear we have seen examples of covariance already. Notice that

$$\mathsf{std}\left(x\right)^2 = \mathsf{var}\left(\{x\}\right) = \mathsf{cov}\left(\{x\},\{x\}\right)$$

which you can prove by substituting the expressions. Recall that variance measures the tendency of a dataset to be different from the mean, so the covariance of a dataset with itself is a measure of its tendency not to be constant.

More important, notice that

$$\mathsf{corr}\left(\{(x,y)\}\right) = \frac{\mathsf{cov}\left(\{x\},\{y\}\right)}{\sqrt{\mathsf{cov}\left(\{x\},\{x\}\right)}\sqrt{\mathsf{cov}\left(\{y\},\{y\}\right)}}.$$

This is occasionally a useful way to think about correlation. It says that the correlation measures the tendency of $\{x\}$ and $\{y\}$ to be larger (resp. smaller) than their

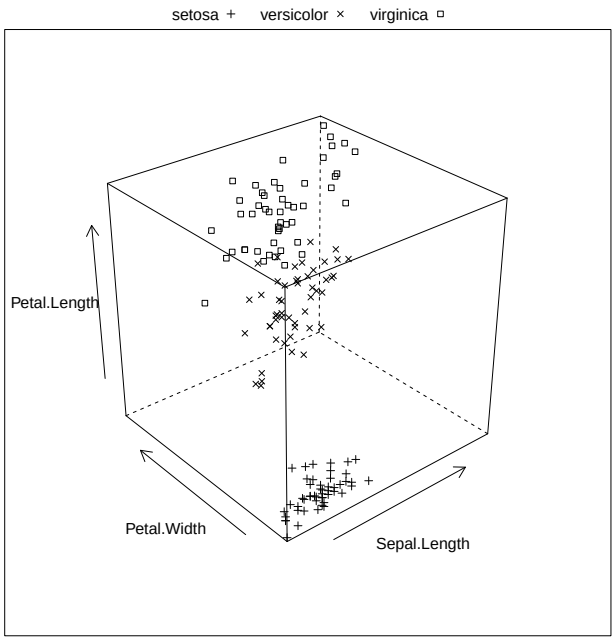


FIGURE 4.6: *This is a 3D scatterplot for the famous Iris data, originally due to ***. I have chosen three variables from the four, and have plotted each species with a different marker. You can see from the plot that the species cluster quite tightly, and are different from one another. R code for this plot is on the website.*

means for the same data points, *compared to* how much they change on their own.

Working with covariance (rather than correlation) allows us to unify some ideas. In particular, for data items which are $d$ dimensional vectors, it is straightforward to compute a single matrix that captures all covariances between all pairs of components — this is the **covariance matrix**.

**Definition: 4.2** *Covariance Matrix*

The covariance matrix is:

$$\mathsf{Covmat}\left(\{\mathbf{x}\}\right) = \frac{\sum_i (\mathbf{x}_i - \mathsf{mean}\left(\{\mathbf{x}\}\right))(\mathbf{x}_i - \mathsf{mean}\left(\{\mathbf{x}\}\right))^T}{N}$$

Notice that it is quite usual to write a covariance matrix as $\Sigma$, and we will follow this convention.

**Properties of the Covariance Matrix** Covariance matrices are often written as $\Sigma$, whatever the dataset (you get to figure out precisely which dataset is

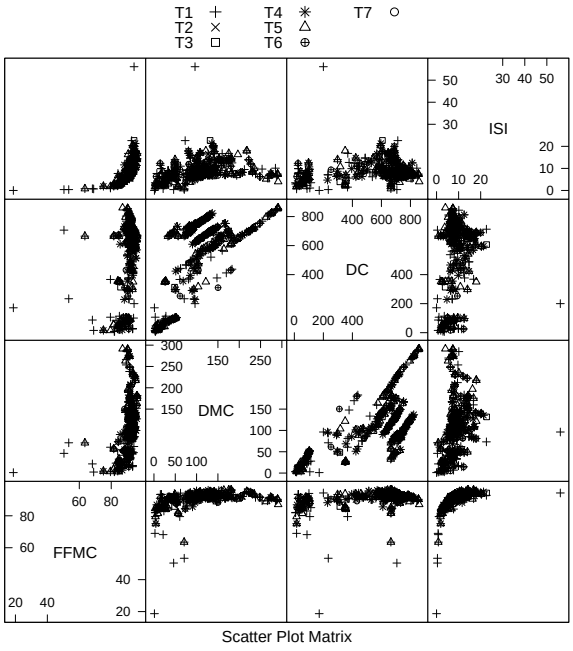


FIGURE 4.7: *This is a scatterplot matrix for the fire dataset from the UC Irvine repository. The smallest area fire is 'T1', and the largest is 'T7'; each is plotted with a different marker. These plots show severity of the fire, plotted against variables 5-8 of the dataset. You should notice that there isn't much separation between the markers. It might be very hard to predict the severity of a fire from these variables. R code for this plot is on the website.*

intended, from context). Generally, when we want to refer to the $j$, $k$'th entry of a matrix $\mathcal{A}$, we will write $\mathcal{A}_{jk}$, so $\Sigma_{jk}$ is the covariance between the $j$'th and $k$'th components of the data.

- The $j$, $k$'th entry of the covariance matrix is the covariance of the $j$'th and the $k$'th components of $\mathbf{x}$, which we write $\mathsf{cov}\left(\left\{x^{(j)}\right\}, \left\{x^{(k)}\right\}\right)$.
- The $j$, $j$'th entry of the covariance matrix is the variance of the $j$'th component of $\mathbf{x}$.
- The covariance matrix is symmetric.
- The covariance matrix is always positive semi-definite; it is positive definite, *unless* there is some vector $\mathbf{a}$ such that $\mathbf{a}^T(\mathbf{x}_i - \mathsf{mean}\left(\{\mathbf{x}_i\}\right) = 0$ for all $i$.

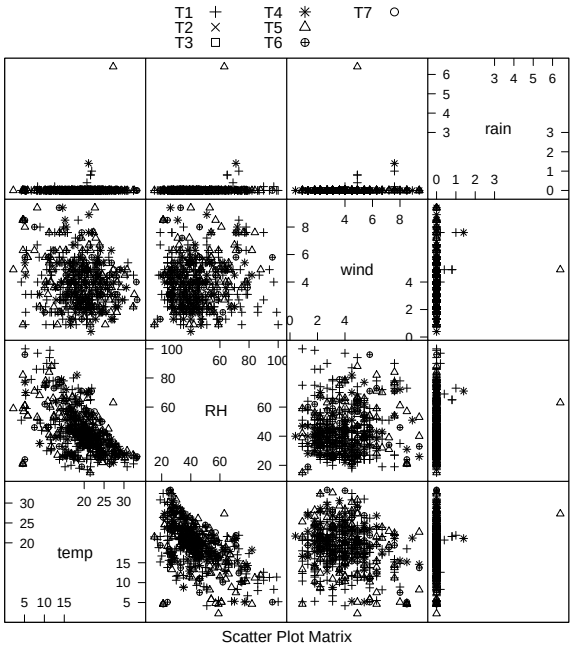


FIGURE 4.8: *This is a scatterplot matrix for the fire dataset from the UC Irvine repository. The smallest area fire is 'T1', and the largest is 'T7'; each is plotted with a different marker. These plots show severity of the fire, plotted against variables 9-12 of the dataset. You should notice that there isn't much separation between the markers. It might be very hard to predict the severity of a fire from these variables. R code for this plot is on the website.*

**Proposition:**

$$\text{Covmat}\left(\{\mathbf{x}\}\right)_{jk} = \text{cov}\left(\left\{x^{(j)}\right\}, \left\{x^{(k)}\right\}\right)$$

**Proof:** Recall

$$\text{Covmat}\left(\{\mathbf{x}\}\right) = \frac{\sum_i (\mathbf{x}_i - \text{mean}\left(\{\mathbf{x}\}\right))(\mathbf{x}_i - \text{mean}\left(\{\mathbf{x}\}\right))^T}{N}$$

and the $j, k$'th entry in this matrix will be

$$\frac{\sum_i (x_i^{(j)} - \text{mean}\left(\left\{x^{(j)}\right\}\right))(x_i^{(k)} - \text{mean}\left(\left\{x^{(k)}\right\}\right))^T}{N}$$

which is $\text{cov}\left(\left\{x^{(j)}\right\}, \left\{x^{(k)}\right\}\right)$.

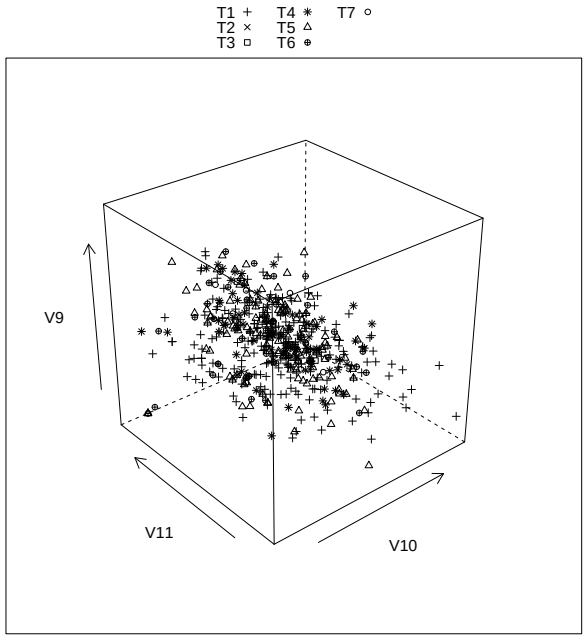


FIGURE 4.9: *This is a 3D scatterplot for the fire dataset from the UC Irvine repository. The smallest area fire is 'T1', and the largest is 'T7'; each is plotted with a different marker. These plots show severity of the fire, plotted against variables 9-11 of the dataset. You should notice that there isn't much separation between the markers. It might be very hard to predict the severity of a fire from these variables. R code for this plot is on the website.*

**Proposition:**

$$\text{Covmat}\left(\{\mathbf{x}_i\}\right)_{jj} = \Sigma_{jj} = \text{var}\left(\left\{x^{(j)}\right\}\right)$$

**Proof:**

$$\begin{aligned}\text{Covmat}\left(\{\mathbf{x}\}\right)_{jj} &= \text{cov}\left(\left\{x^{(j)}\right\}, \left\{x^{(j)}\right\}\right) \\ &= \text{var}\left(\left\{x^{(j)}\right\}\right)\end{aligned}$$

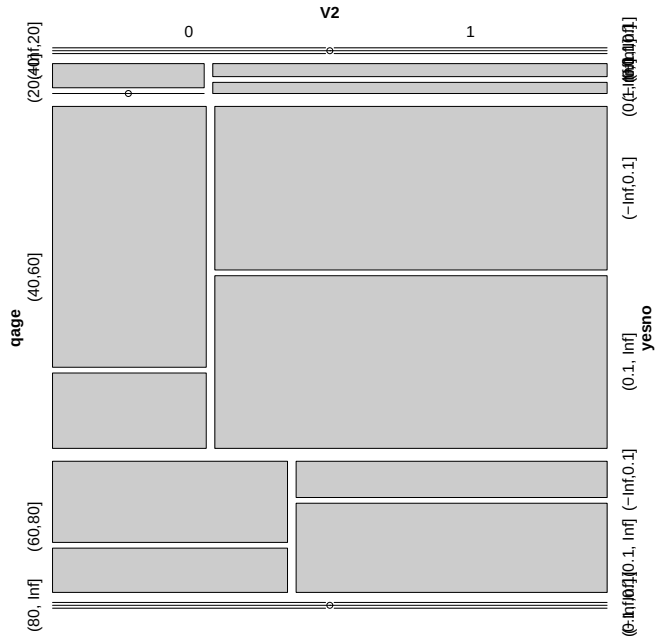


FIGURE 4.10: *This is a mosaic plot of age, gender, and level of disease for the heart dataset from the UC Irvine repository. Notice that the data consists mainly of people aged 40-60. There are very few people younger than 20 or older than 80. A significantly greater percentage of the measurements comes from the gender labelled 1, and in all age groups the percentage of that gender that has the disease level is higher. This suggests that this gender is male. Notice also that the percentage of males with the disease is really quite high (at least 50% in each case). This suggests that either the population is special in some way — perhaps the measurements are collected from people who are feeling sick — or that the criterion used to determine whether an individual is diseased is too sensitive. R code for this plot is on the website.*

**Proposition:**

$$\mathsf{Covmat}\left(\{\mathbf{x}\}\right) = \mathsf{Covmat}\left(\{\mathbf{x}\}\right)^T$$

**Proof:** We have

$$\begin{aligned}\mathsf{Covmat}\left(\{\mathbf{x}\}\right)_{jk} &= \mathsf{cov}\left(\left\{x^{(j)}\right\}, \left\{x^{(k)}\right\}\right) \\ &= \mathsf{cov}\left(\left\{x^{(k)}\right\}, \left\{x^{(j)}\right\}\right) \\ &= \mathsf{Covmat}\left(\{\mathbf{x}\}\right)_{kj}\end{aligned}$$

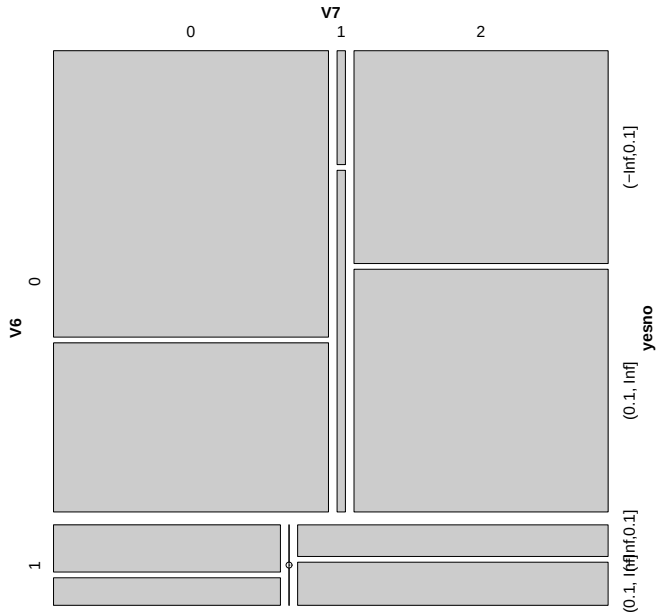


FIGURE 4.11: *This is a mosaic plot of variables six (V6) and seven (V7) and level of disease for the heart dataset from the UC Irvine repository. These variables represent some physiological properties of importance, but I don't know their interpretation. V6 does not seem to be particularly significant, but the population for which V7 has value 1 has a high incidence of disease. R code for this plot is on the website.*

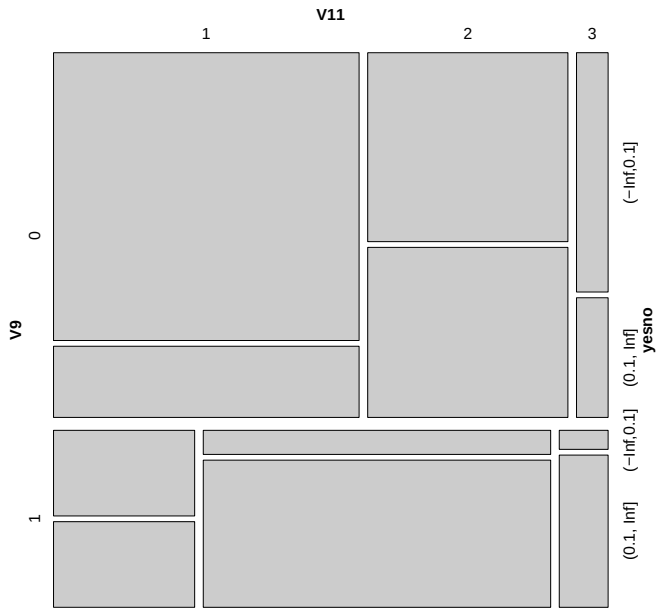


FIGURE 4.12: *This is a mosaic plot of variables nine (V9) and eleven (V11) and level of disease for the heart dataset from the UC Irvine repository. These variables represent some physiological properties of importance, but I don't know their interpretation. V9 at level one is clearly not a good thing. If V9 is at level 0, then V11 at level 1 is also a problem. R code for this plot is on the website.*

**Proposition:** *Write* $\Sigma = \mathsf{Covmat}(\{\mathbf{x}\})$. *If there is no vector* $\mathbf{a}$ *such that* $\mathbf{a}^T(\mathbf{x}_i - \mathsf{mean}(\{\mathbf{x}\})) = 0$ *for all* $i$, *then for any vector* $\mathbf{u}$, *such that* $\|\mathbf{u}\| > 0$,

$$\mathbf{u}^T\Sigma\mathbf{u} > 0.$$

*If there is such a vector* $\mathbf{a}$, *then*

$$\mathbf{u}^T\Sigma\mathbf{u} \geq 0.$$

**Proof:** We have

$$\begin{aligned}\mathbf{u}^T\Sigma\mathbf{u} &= \frac{1}{N}\sum_i \left[\mathbf{u}^T(\mathbf{x}_i - \mathsf{mean}(\{\mathbf{x}\}))\right]\left[(\mathbf{x}_i - \mathsf{mean}(\{\mathbf{x}\}))^T\mathbf{u}\right] \\ &= \frac{1}{N}\sum_i \left[\mathbf{u}^T(\mathbf{x}_i - \mathsf{mean}(\{\mathbf{x}\}))\right]^2.\end{aligned}$$

Now this is a sum of squares. If there is some $\mathbf{a}$ such that $\mathbf{a}^T(\mathbf{x}_i - \mathsf{mean}(\{\mathbf{x}\})) = 0$ for every $i$, then the covariance matrix must be positive semidefinite (because the sum of squares could be zero in this case). Otherwise, it is positive definite, because the sum of squares will always be positive.

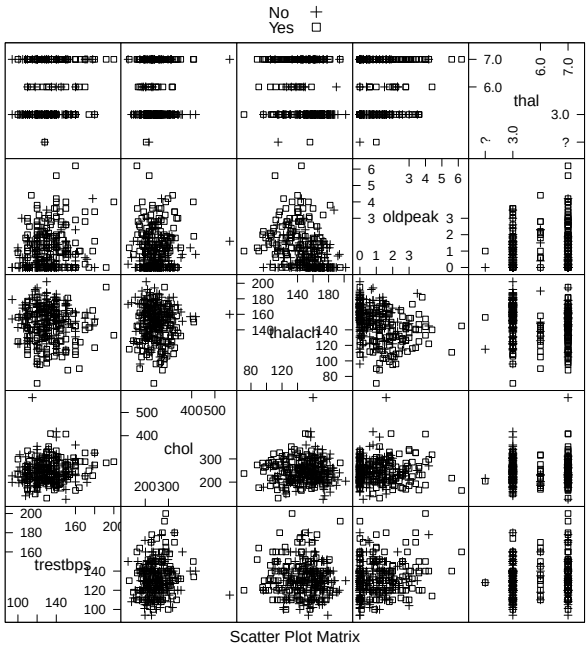


FIGURE 4.13: *This is a scatterplot matrix of five variables for the heart dataset from the UC Irvine repository. The shape of the marker indicates disease or not. The variable names are taken from the description at that URL. "trestbps" is a measurement of resting systolic blood pressure, and "chol" of cholesterol (I don't know which lipid, or in which units). You should notice that "thal" takes only a discrete set of values. Notice also that it appears to be unwise to have very large values of "trestbps", "chol", or "oldpeak" (or smal values of "thalach"), it isn't that easy to distinguish between the different cases. There isn't a clear clustering the way there was in the iris data. R code for this plot is on the website.*

## 4.2 BLOB ANALYSIS OF HIGH-DIMENSIONAL DATA

When we plotted histograms, we saw that mean and variance were a very helpful description of data that had a unimodal histogram. If the histogram had more than one mode, one needed to be somewhat careful to interpret the mean and variance; in the pizza example, we plotted diameters for different manufacturers to try and see the data as a collection of unimodal histograms.

Generally, mean and covariance are a good description of data that lies in a "blob" (Figure 4.14). You might not believe that this is a technical term, but it's quite widely used. This is because mean and covariance supply a natural coordinate system in which to interpret the blob. Mean and covariance are less useful as descriptions of data that forms multiple blobs (Figure 4.14). In chapter 1, we discuss methods to model data that forms multiple blobs, or other shapes that we

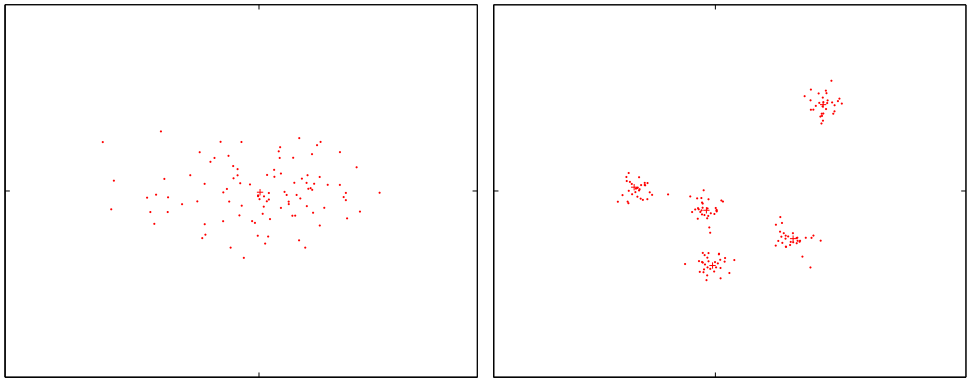

FIGURE 4.14: *On the* **left**, *a "blob" in two dimensions. This is a set of data points that lie somewhat clustered around a single center, given by the mean. I have plotted the mean of these data points with a '+'. On the* **right**, *a data set that is best thought of as a collection of five blobs. I have plotted the mean of each with a '+'. We could compute the mean and covariance of this data, but it would be less revealing than the mean and covariance of a single blob. In chapter 1, I will describe automatic methods to describe this dataset as a series of blobs.*

will interpret as a set of blobs. But many datasets really are single blobs, and we concentrate on such data here. The way to understand a blob is to think about the coordinate transformations that place a blob into a particularly convenient form.

### 4.2.1 Transforming High Dimensional Data

Assume we apply an affine transformation to our data set $\{\mathbf{x}\}$, to obtain a new dataset $\{\mathbf{u}\}$, where $\mathbf{u}_i = \mathcal{A}\mathbf{x}_i + \mathbf{b}$. Here $\mathcal{A}$ is any matrix (it doesn't have to be square, or symmetric, or anything else; it just has to have second dimension $d$). It is easy to compute the mean and covariance of $\{\mathbf{u}\}$. We have

$$\begin{aligned} \text{mean}\left(\{\mathbf{u}\}\right) &= \text{mean}\left(\{\mathcal{A}\mathbf{x} + \mathbf{b}\}\right) \\ &= \mathcal{A}\text{mean}\left(\{\mathbf{x}\}\right) + \mathbf{b}, \end{aligned}$$

so you get the new mean by multiplying the original mean by $\mathcal{A}$ and adding $\mathbf{b}$.

The new covariance matrix is easy to compute as well. We have:

$$\begin{aligned} \text{Covmat}\left(\{\mathbf{u}\}\right) &= \text{Covmat}\left(\{\mathcal{A}\mathbf{x} + \mathbf{b}\}\right) \\ &= \frac{\sum_i (\mathbf{u}_i - \text{mean}\left(\{\mathbf{u}\}\right))(\mathbf{u}_i - \text{mean}\left(\{\mathbf{u}\}\right))^T}{N} \\ &= \frac{\sum_i (\mathcal{A}\mathbf{x}_i + \mathbf{b} - \mathcal{A}\text{mean}\left(\{\mathbf{x}\}\right) - \mathbf{b})(\mathcal{A}\mathbf{x}_i + \mathbf{b} - \mathcal{A}\text{mean}\left(\{\mathbf{x}\}\right) - \mathbf{b})^T}{N} \\ &= \frac{\mathcal{A}\sum_i (\mathbf{x}_i - \text{mean}\left(\{\mathbf{x}\}\right))(\mathbf{x}_i - \text{mean}\left(\{\mathbf{x}\}\right))^T \mathcal{A}^T}{N} \\ &= \mathcal{A}\text{Covmat}\left(\{\mathbf{x}\}\right)\mathcal{A}^T. \end{aligned}$$

CHAPTER 5

# Learning to Classify

A **classifier** is a procedure that accepts a set of features and produces a class label for them. There could be two, or many, classes, though it is usual to produce multi-class classifiers out of two-class classifiers. Classifiers are immensely useful, and find wide application, because many problems are naturally decision problems. For example, if you wish to determine whether to place an advert on a web-page or not, you would use a classifier (i.e. look at the page, and say yes or no according to some rule). As another example, if you have a program that you found for free on the web, you would use a classifier to decide whether it was safe to run it (i.e. look at the program, and say yes or no according to some rule). As yet another example, you can think of doctors as extremely complex multi-class classifiers.

Classifiers are built by taking a set of labeled examples and using them to come up with a rule that assigns a label to any new example. In the general problem, we have a training dataset $(\mathbf{x}_i, y_i)$; each of the **feature vectors** $\mathbf{x}_i$ consists of measurements of the properties of different types of object, and the $y_i$ are labels giving the type of the object that generated the example.

## 5.1 CLASSIFICATION, ERROR, AND LOSS

You should think of a classifier as a rule, though it might not be implemented that way. We pass in a feature vector, and the rule returns a class label. We know the relative costs of mislabeling each class and must come up with a rule that can take any plausible $\mathbf{x}$ and assign a class to it, in such a way that the expected mislabeling cost is as small as possible, or at least tolerable. For most of this chapter, we will assume that there are two classes, labeled 1 and $-1$. Section 5.4.2 shows methods for building multi-class classifiers from two-class classifiers.

### 5.1.1 Using Loss to Determine Decisions

The choice of classification rule must depend on the cost of making a mistake. A two-class classifier can make two kinds of mistake. A **false positive** occurs when a negative example is classified positive; a **false negative** occurs when a positive example is classified negative. For example, pretend there is only one disease; then doctors would be classifiers, deciding whether a patient had it or not. If this disease is dangerous, but is safely and easily treated, then false negatives are expensive errors, but false positives are cheap. Similarly, if it is not dangerous, but the treatment is difficult and unpleasant, then false positives are expensive errors and false negatives are cheap.

### 5.1.2 Training Error, Test Error, and Overfitting

It can be quite difficult to know a good loss function, but one can usually come up with a plausible model. If we knew the posterior probabilities, building a classifier would be straightforward. Usually we don't, and must build a model from data. This model could be a model of the posterior probabilities, or an estimate of the decision boundaries. In either case, we have only the training data to build it with. **Training error** is the error a model makes on the training data set.

Generally, we will try to make this training error small. However, what we really want to minimize is the **test error**, the error the classifier makes on test data. We cannot minimize this error directly, because we don't know the test set (if we did, special procedures in training apply). However, classifiers that have small training error might not have small test error. One example of this problem is the (silly) classifier that takes any data point and, if it is the same as a point in the training set, emits the class of that point and otherwise chooses randomly between the classes. This classifier has been learned from data, and has a zero error rate on the training dataset; it is likely to be unhelpful on any other dataset, however.

The phenomenon that causes test error to be worse than training error is sometimes called **overfitting** (other names include **selection bias**, because the training data has been selected and so isn't exactly like the test data, and **generalizing badly**, because the classifier fails to generalize). It occurs because the classifier has been trained to perform well *on the training dataset*. The training dataset is not the same as the test dataset. First, it is quite likely smaller. Second, it might be biased through a variety of accidents. This means that small training error may have to do with quirks of the training dataset that don't occur in other sets of examples. It is quite possible that, in this case, the test error will be larger than the training error. Generally, we expect classifiers to perform somewhat better on the training set than on the test set. Overfitting can result in a substantial difference between performance on the training set and performance on the test set. One consequence of overfitting is that classifiers should always be evaluated on test data. Doing this creates other problems, which we discuss in Section 5.1.3.

A procedure called **regularization** attaches a penalty term to the training error to get a better estimate of the test error. This penalty term could take a variety of different forms, depending on the requirements of the application. Section 1 describes regularization in further detail.

### 5.1.3 Error Rate and Cross-Validation

There are a variety of methods to describe the performance of a classifier. Natural, straightforward choices are to report the **error rate**, the percentage of classification attempts on a test set that result in the wrong answer. This presents an important difficulty. We cannot estimate the error rate of the classifier using training data, because the classifier has been trained to do well on that data, which will mean our error rate estimate will be an underestimate. An alternative is to separate out some training data to form a **validation set**, then train the classifier on the rest of the data, and evaluate on the validation set. This has the difficulty that the classifier will not be the best estimate possible, because we have left out some training data

when we trained it. This issue can become a significant nuisance when we are trying to tell which of a set of classifiers to use—did the classifier perform poorly on validation data because it is not suited to the problem representation or because it was trained on too little data?

We can resolve this problem with **cross-validation**, which involves repeatedly: splitting data into training and validation sets uniformly and at random, training a classifier on the training set, evaluating it on the validation set, and then averaging the error over all splits. This allows an estimate of the likely future performance of a classifier, at the expense of substantial computation.

Choose some class of subsets of the training set,
for example, singletons.

For each element of that class, construct a classifier by
omitting that element in training, and compute the
mean number of classification errors on the omitted subset.

Average these errors over the class of subsets to estimate
the risk of using the classifier trained on the entire training
dataset.

**Algorithm 5.1:** *Cross-Validation*

The most usual form of this algorithm involves omitting single items from the dataset and is known as **leave-one-out cross-validation**. Errors are usually estimated by simply averaging over the class, but more sophisticated estimates are available. We do not justify this tool mathematically; however, it is worth noticing that leave-one-out cross-validation, in some sense, looks at the sensitivity of the classifier to a small change in the training set. If a classifier performs well under this test, then large subsets of the dataset look similar to one another, which suggests that a representation of the relevant probabilities derived from the dataset might be quite good.

## 5.2 LINEAR CLASSIFIERS

Assume we have a set of $N$ example points $\mathbf{x}_i$ that belong to two classes, which we indicate by 1 and $-1$. These points come with their class labels, which we write as $y_i$; thus, our dataset can be written as

$$\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}.$$

We wish to the sign of $y$ for any point $\mathbf{x}$; this rule is our classifier. Write $y_i^{(p)}(\mathbf{x})$ for the predicted value of $y$ for a given value of $\mathbf{x}$. assume we have a set of $N$ example points $\mathbf{x}_i$ that belong to two classes, which we indicate by 1 and $-1$. These points come with their class labels, which we write as $y_i$; thus, our dataset can be written as

$$\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}.$$

We seek a rule that predicts the sign of $y$ for any point $\mathbf{x}$; this rule is our classifier. We will use a linear rule, so that for a new data item $\mathbf{x}$, we will predict

$$\text{sign}\left((\mathbf{w} \cdot \mathbf{x} + b)\right).$$

You should think of $\mathbf{w}$ and $b$ as representing a hyperplane which separates the positive data from the negative data. This hyperplane is known as the **decision boundary**. The particular rule is given by the choice of $\mathbf{w}$ and $b$.

### 5.2.1 Why a linear rule?

This family of rules may look bad to you. It is easy to come up with examples that it misclassifies badly. The rule has important strengths: it is easy to estimate the best choice of rule, it works very well in practice on real data, and it is fast to evaluate. For practical examples, experience shows that the error rate can be improved by adding features to the vector $\mathbf{x}$.

**Example: 5.1** *A linear model with a single feature*

Assume we use a linear model with one feature. Then the model has the form $y_i^{(p)} = \text{sign}(ax_i + b)$. For any particular example which has the feature value $x^*$, this means we will test whether $x^*$ is larger than, or smaller than, $-b/a$.

**Example: 5.2** *A linear model with two features*

Assume we use a linear model with two features. Then the model has the form $y_i^{(p)} = \text{sign}(\mathbf{a}^T\mathbf{x}_i + b)$. The sign changes along the line $\mathbf{a}^T\mathbf{x} + b = 0$. You should check that this is, indeed, a line. On one side of this line, the model makes positive predictions; on the other, negative. Which side is which can be swapped by multiplying $\mathbf{a}$ and $b$ by $-1$.

### 5.2.2 Logistic Regression

We will choose $\mathbf{a}$ and $b$ by choosing values that minimize the cost of errors made by the classifier. In particular, we will adopt a cost function of the form:

$$\text{Training error cost} + \text{penalty term}.$$

For the moment, we will ignore the penalty term. The training error cost will be of the form

$$(1/N)\sum_{i=1}^{N} C((\mathbf{a}^T\mathbf{x}_i + b), y_i)$$

so at each point in the training data, we compute a cost from the true value of $y_i$ and the predicted value. This cost should be large if $y_i$ and $y_i^{(p)}(\mathbf{a}^T\mathbf{x}_i + b, y_i)$ have different signs, and small if they have the same sign. It is convenient to write

$$\gamma_i = (\mathbf{a}^T\mathbf{x}_i + b, y_i).$$

For **logistic regression**, the cost function using this notation is

$$C((\mathbf{a}^T\mathbf{x}_i + b), y_i) = C(\gamma_i, y_i) = \log\left(1 + \exp\left(-y_i\gamma_i\right)\right).$$

The function $L(1, \gamma)$ is plotted in Figure 5.1. This loss is sometimes known as the **logistic loss**. This loss very strongly penalizes a large positive $\gamma_i$ if $y_i$ is negative (and vice versa). However, there is no significant advantage to having a large positive $\gamma_i$ if $y_i$ is positive. This means that the significant components of the loss function will be due to examples that the classifier gets wrong, but also due to examples that have $\gamma_i$ near zero (i.e., the example is close to the decision boundary).

You should notice another important property of this loss. Assume we wish to predict a label for a new data item. The loss we would incur depends quite strongly on the magnitude of $\gamma$. If we produce a large value of $\gamma$ for that data item *with the wrong sign*, then we would incur a very large loss. This means that we should prefer values of $\mathbf{a}$ and $b$ that will tend to produce small values of $\gamma$. In turn, we should prefer small values of $\mathbf{a}$ if they give about the same value of training loss. This is our penalty term. We should use a cost function of the form

$$\text{Training Loss} + \frac{\lambda}{2}\left(\text{Norm of } \mathbf{a}\right)$$

which is

$$\left[\frac{1}{N}\sum_{i \in \text{examples}} \{\log(1 + \exp -y_i\gamma_i\})\right] + \frac{\lambda}{2}\mathbf{a}^T\mathbf{a}$$

where $\lambda > 0$ is a constant chosen for good performance. Too large a value of $\lambda$, and the classifier will behave poorly on training and test data; too small a value, and the classifier will behave poorly on test data.

Usually, the value of $\lambda$ is set with a validation dataset. We train classifiers with different values of $\lambda$ on a test dataset, then evaluate them on a validation set—data whose labels are known, but which is not used for training—and finally choose the $\lambda$ that gets the best validation error. The error is not usually all that sensitive to the choice of $\lambda$, so searching values by factors of 10 is usually fine.

The penalty term is often referred to as a **regularizer**, because it tends to discourage solutions that are large (and so have possible high loss on future test data) but are not strongly supported by the training data.

### 5.2.3 The Hinge Loss

There are alternate losses, that are very useful. The linear **support vector machine** or **SVM** uses the **hinge loss**. In this case the loss comparing the label value $y_i$ and the prediction $\gamma_i = (\mathbf{w} \cdot \mathbf{x}_i + b)$ can be written as

$$L_h(y_i, \gamma_i) = \max(0, 1 - y_i\gamma_i).$$

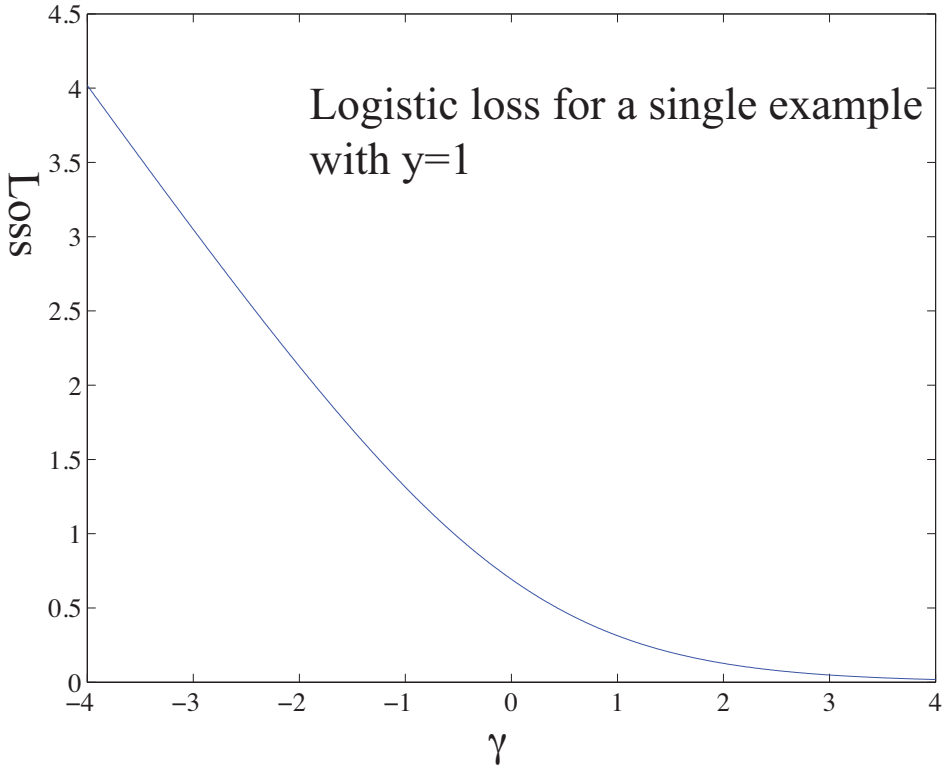


FIGURE 5.1: *The logistic loss, plotted for the case $y_i = 1$. In the case of the logistic loss, the horizontal variable is the $\gamma_i = \mathbf{a} \cdot \mathbf{x}_i$ of the text. Notice that giving a strong negative response to this positive example causes a loss that grows linearly as the magnitude of the response grows. Notice also that giving an insufficiently positive response also causes a loss. Giving a strongly positive response is cheap or free.*

This loss is always non-negative. For the moment, assume $y_i = 1$; then, any prediction by the classifier with value greater than one will incur no loss, and any smaller prediction will incur a cost that is linear in the prediction value (Figure **??**). This means that minimizing the loss will encourage the classifier to (a) make strong positive (or negative) predictions for positive (or negative) examples and (b) for examples it gets wrong, make the most positive (negative) prediction that it can. The expression

$$\left[(1/N)\sum_{i=1}^{N}\max(0, 1 - y_i\left(\mathbf{w}\cdot\mathbf{x}_i + b\right))\right] + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

fits into the rule of above, where we obtained a classifier by minimizing

$$\text{Training Loss} + \text{Regularizer}.$$

We have just changed the loss.

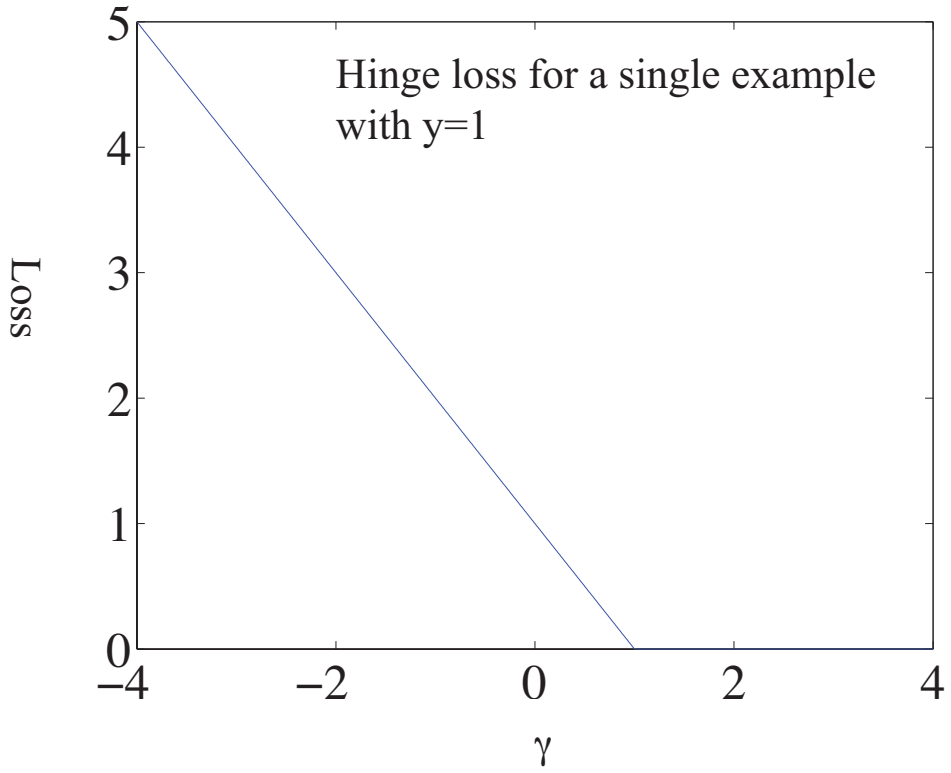


FIGURE 5.2: *The hinge loss, plotted for the case $y_i = 1$. The horizontal variable is the $\gamma_i = \mathbf{a} \cdot \mathbf{x}_i$ of the text. Notice that giving a strong negative response to this positive example causes a loss that grows linearly as the magnitude of the response grows. Notice also that giving an insufficiently positive response also causes a loss. Giving a strongly positive response is or free. The loss should look a lot like the hinge loss to you.*

## 5.3 BASIC IDEAS FOR NUMERICAL MINIMIZATION

We must now obtain a classifier that minimizes either logistic or hinge loss. Assume we have a function $g(\mathbf{a})$, and we wish to obtain a value of $\mathbf{a}$ that achieves the minimum for that function. Sometimes we can solve this problem in closed form by constructing the gradient and finding a value of $\mathbf{a}$ the makes the gradient zero. More usually we need a numerical method. Implementing these numerical methods is a specialized business, and it is usual to use general optimization codes. This section is intended to sketch how such codes work, so you can read manual pages, etc. more effectively. Personally, I am a happy user of Matlab's `fminunc`, although the many different settings take some getting used to.

### 5.3.1 Overview

Typical codes take a description of the objective function (typically, the name of a function), a start point for the search, and a collection of parameters. All codes take an estimate $\mathbf{a}^{(i)}$, update it to $\mathbf{a}^{(i+1)}$, then check to see whether the result is a minimum. This process is started from the start point. The update is usually obtained by computing a direction $\mathbf{p}^{(i)}$ such that for small values of $h$, $g(\mathbf{a}^{(i)}+h\mathbf{p}^{(i)})$ is smaller than $g(\mathbf{a}^{(i)})$. Such a direction is known as a **descent direction**.

Assume we have a descent direction. We must now choose how far to travel along that direction. We can see $g(\mathbf{a}^{(i)}+h\mathbf{p}^{(i)})$ as a function of $h$. Write this function as $\phi(h)$. We start at $h = 0$ (which is the original value $\mathbf{a}^{(i)}$, so $\phi(0) = g(\mathbf{a}^{(i)})$, and move in the direction of increasing $h$ to find a small value of $\phi(h)$ that is less than $\phi(0)$. The descent direction was chosen so that for small $h > 0$, $\phi(h) < \phi(0)$; one way to tell we are at a minimum is we cannot choose a descent direction. Searching for a good value of $h$ is known as **line search**. Typically, this search involves a sequence of estimated values of $h$, which we write $h_i$. One algorithm is to start with (say) $h_0 = 1$; if $\phi(h_i)$ is not small enough (and there are other tests we may need to apply — this is a summary!), we compute $h_{(i+1)} = (1/2)h_i$. This stops when some $h_i$ passes a test, or when it is so small that the step is pointless.

### 5.3.2 Gradient Descent

One method to choose a descent direction is **gradient descent**, which uses the negative gradient of the function. Recall our notation that

$$\mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ \dots \\ a_d \end{pmatrix}$$

and that

$$\nabla g = \begin{pmatrix} \frac{\partial g}{\partial a_1} \\ \frac{\partial g}{\partial a_2} \\ \dots \\ \frac{\partial g}{\partial a_d} \end{pmatrix}.$$

We can write a Taylor series expansion for the function $g(\mathbf{a}^{(i)} + h\mathbf{p}^{(i)})$. We have that

$$g(\mathbf{a}^{(i)} + h\mathbf{p}^{(i)}) = g(\mathbf{a}^{(i)}) + h(\nabla g)^T\mathbf{p}^{(i)} + O(h^2)$$

This means that we can expect that if

$$\mathbf{p}^{(i)} = -\nabla g(\mathbf{a}^{(i)}),$$

we expect that, at least for small values of $h$, $g(\mathbf{a}^{(i)}+h\mathbf{p}^{(i)})$ will be less than $g(\mathbf{a}^{(i)})$.

This works (as long as $g$ is differentiable, and quite often when it isn't) because $g$ must go down for at least small steps in this direction. There are two ways to evaluate a gradient. You can require that the software estimate a numerical derivative for you, which usually slows things down somewhat, or you can supply a

gradient value. Usually this gradient value must be computed by the same function that computes the objective function value.

One tip: in my experience, about 99% of problems with numerical optimization codes occur because the user didn't check that the gradient their function computed is right. Most codes will compute a numerical gradient for you, then check that against your gradient; if they're sufficiently different, the code will complain. You don't want to do this at runtime, because it slows things up, but it's an excellent idea to check.

### 5.3.3 Stochastic Gradient Descent

Assume we wish to minimize some function $g(\mathbf{a}) = g_0(\mathbf{a}) + (1/N)\sum_{i=1}^{N} g_i(\mathbf{a})$, as a function of $\mathbf{a}$. Gradient descent would require us to form

$$-\nabla g(\mathbf{a}) = -\left(\nabla g_0(\mathbf{a}) + (1/N)\sum_{i=1}^{N} \nabla g_i(\mathbf{a})\right)$$

and then take a small step in this direction. But if $N$ is large, this is unattractive, as we might have to sum a lot of terms. This happens a lot in building classifiers, where you might quite reasonably expect to deal with millions of examples. Touching each example at each step really is impractical.

Instead, assume that, at each step, we choose a number $k$ in the range $1 \ldots N$ uniformly and at random, and form

$$\mathbf{p}_k = -\left(\nabla g_0(\mathbf{a}) + \nabla g_k(\mathbf{a})\right)$$

and then take a small step along $\mathbf{p}_k$. Our new point becomes

$$\mathbf{a}^{(i+1)} = \mathbf{a}^{(i)} + \eta \mathbf{p}_k^{(i)},$$

where $\eta$ is called the **steplength** (even though it very often isn't the length of the step we take!). It is easy to show that

$$\mathbb{E}[\mathbf{p}_k] = \nabla g(\mathbf{a})$$

(where the expectation is over the random choice of $k$). This implies that if we take many small steps along $\mathbf{p}_k$, they should average out to a step backwards along the gradient. This approach is known as **stochastic gradient descent** (because we're not going along the gradient, but along a random vector which is the gradient only in expectation). It isn't obvious that stochastic gradient descent is a good idea. Although each step is easy to take, we may need to take more steps. The question is then whether we gain in the increased speed of the step what we lose by having to take more steps. Not much is known theoretically, but in practice the approach is hugely successful for training classifiers.

Choosing a steplength $\eta$ takes some work. Line search won't work, because we don't want to evaluate the function $g$, because doing so involves looking at each of the $g_i$ terms. Instead, one uses a steplength that is large at the start — so that it can explore large changes in the values of the classifier parameters — and small steps later — so that it settles down. One useful strategy is to divide training into

**epochs**. Each epoch is a block of a fixed number of iterations. Each iteration is one of the steps given above, with fixed steplength. However, the steplength changes from epoch to epoch. In particular, in the $r$'th epoch, the steplength is

$$\eta^{(r)} = \frac{a}{r+b}$$

where $a$ and $b$ are constants chosen by experiment with small subsets of the dataset.

One cannot really test whether stochastic gradient descent has converged to the right answer. A better approach is to plot the error as a function of epoch on a validation set. This should vary randomly, but generally go down as the epochs proceed.

### 5.3.4 Example: Training a Support Vector Machine with Stochastic Gradient Descent

We need to choose $\mathbf{w}$ and $b$ to minimize

$$C(\mathbf{w}, b) = (1/N) \sum_{i=1}^{N} \max(0, 1 - y_i (\mathbf{w} \cdot \mathbf{x}_i + b)) + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w}.$$

There are several methods to do so. Section 1 describes some of the many available support vector machine training packages on the web; it is often, even usually, a good idea to use one of these. But it is worth understanding how such things work.

For a support vector machine, stochastic gradient descent is particularly easy. We have estimates $\mathbf{w}^{(n)}$ and $b^{(n)}$ of the classifier parameters, and we want to improve the estimates. We pick the $k$'th example at random. We must now compute

$$\nabla \left( \max(0, 1 - y_k (\mathbf{w} \cdot \mathbf{x}_k + b)) + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w} \right).$$

Assume that $y_k (\mathbf{w} \cdot \mathbf{x}_k + b) > 1$. In this case, the classifier predicts a score with the right sign, and a magnitude that is greater than one. Then the first term is zero, and the gradient of the second term is easy. Now if $y_k (\mathbf{w} \cdot \mathbf{x}_k + b) < 1$, we can ignore the max, and the first term is $1 - y_k (\mathbf{w} \cdot \mathbf{x}_k + b)$; the gradient is again easy. But what if $y_k (\mathbf{w} \cdot \mathbf{x}_k + b) = 1$? there are two distinct values we could choose for the gradient, because the max term isn't differentiable. It turns out not to matter which term we choose (Figure **??**), so we can write the gradient as

$$p_k = \begin{cases} \begin{bmatrix} \lambda \mathbf{w} \\ 0 \end{bmatrix} & \text{if } y_k (\mathbf{w} \cdot \mathbf{x}_k + b) \geq 1 \\ \begin{bmatrix} \lambda \mathbf{w} - y_k \mathbf{x} \\ -y_k \end{bmatrix} & \text{otherwise} \end{cases}$$

We choose a steplength $\eta$, and update our estimates using this gradient. This yields:

$$\mathbf{w}^{(n+1)} = \mathbf{w}^{(n)} - \eta \begin{cases} \lambda \mathbf{w} & \text{if } y_k (\mathbf{w} \cdot \mathbf{x}_k + b) \geq 1 \\ \lambda \mathbf{w} - y_k \mathbf{x} & \text{otherwise} \end{cases}$$

and

$$b^{(n+1)} = b^{(n)} - \eta \begin{cases} 0 & \text{if } y_k (\mathbf{w} \cdot \mathbf{x}_k + b) \geq 1 \\ -y_k & \text{otherwise} \end{cases}.$$

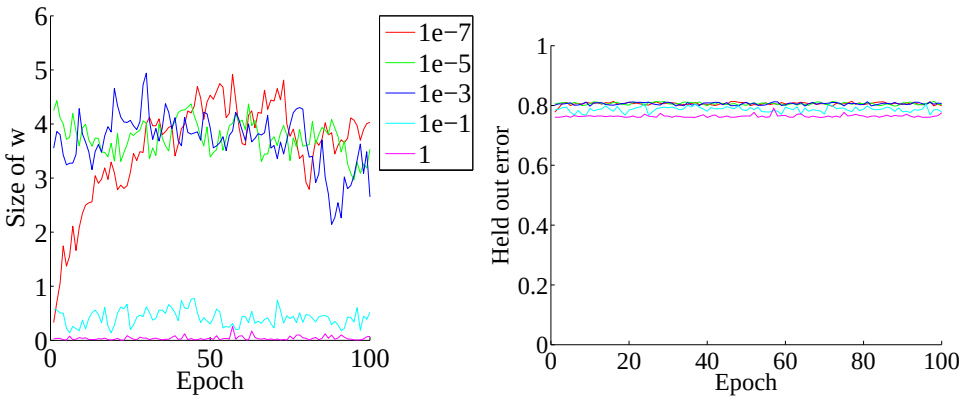


FIGURE 5.3: *On the* **left**, *the magnitude of the weight vector* $\mathbf{w}$ *at the end of each epoch for the first training regime described in the text. On the* **right**, *the accuracy on held out data at the end of each epoch. Notice how different choices of regularization parameter lead to different magnitudes of* $\mathbf{w}$*; how the method isn't particularly sensitive to choice of regularization parameter (they change by factors of 100); how the accuracy settles down fairly quickly; and how overlarge values of the regularization parameter do lead to a loss of accuracy.*

To construct figures, I downloaded the dataset at `http://archive.ics.uci.edu/ml/datasets/Adult`. This dataset apparently contains 48, 842 data items, but I worked with only the first 32, 000. Each consists of a set of numeric and categorical features describing a person, together with whether their annual income is larger than or smaller than 50K\$. I ignored the categorical features to prepare these figures. This isn't wise if you want a good classifier, but it's fine for an example. I used these features to predict whether income is over or under 50K\$. I split the data into 5, 000 test examples, and 27,000 training examples. It's important to do so at random. There are 6 numerical features. I subtracted the mean (which doesn't usually make much difference) and rescaled each so that the variance was 1 (which is often very important). I used two different training regimes.

In the first training regime, there were 100 epochs. In each epoch, I applied 426 steps. For each step, I selected one data item uniformly at random (sampling with replacement), then stepped down the gradient. This means the method sees a total of 42, 600 data items. This means that there is a high probability it has touched each data item once (27, 000 isn't enough, because we are sampling with replacement, so some items get seen more than once). I chose 5 different values for the regularization parameter and trained with a steplength of $1/(0.01 * e + 50)$, where $e$ is the epoch. At the end of each epoch, I computed $\mathbf{w}^T\mathbf{w}$ and the accuracy (fraction of examples correctly classified) of the current classifier on the held out test examples. Figure 5.3 shows the results. You should notice that the accuracy changes slightly each epoch; that for larger regularizer values $\mathbf{w}^T\mathbf{w}$ is smaller; and that the accuracy settles down to about 0.8 very quickly.

In the second training regime, there were 100 epochs. In each epoch, I applied

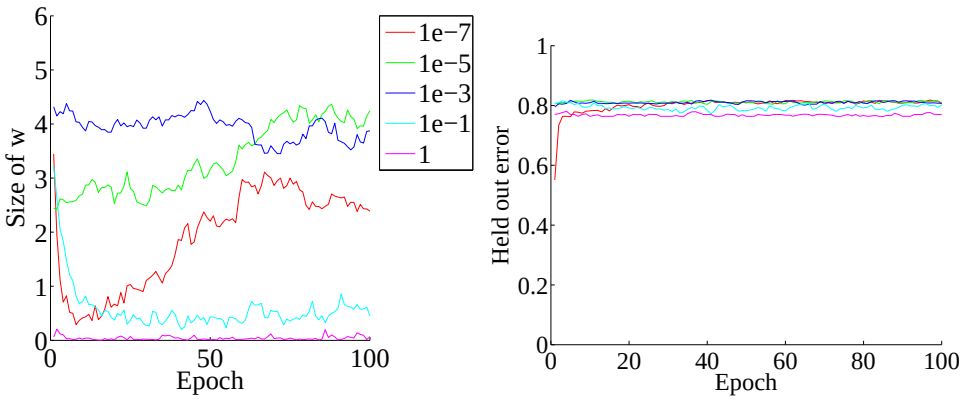


FIGURE 5.4: *On the* **left**, *the magnitude of the weight vector* $\mathbf{w}$ *at the end of each epoch for the second training regime described in the text. On the* **right**, *the accuracy on held out data at the end of each epoch. Notice how different choices of regularization parameter lead to different magnitudes of* $\mathbf{w}$*; how the method isn't particularly sensitive to choice of regularization parameter (they change by factors of 100); how the accuracy settles down fairly quickly; and how overlarge values of the regularization parameter do lead to a loss of accuracy.*

50 steps. For each step, I selected one data item uniformly at random (sampling with replacement), then stepped down the gradient. This means the method sees a total of 5,000 data items, and about 3, 216 unique data items — it hasn't seen the whole training set. I chose 5 different values for the regularization parameter and trained with a steplength of $1/(0.01 * e + 50)$, where $e$ is the epoch. At the end of each epoch, I computed $\mathbf{w}^T\mathbf{w}$ and the accuracy (fraction of examples correctly classified) of the current classifier on the held out test examples. Figure 5.4 shows the results. You should notice that the accuracy changes slightly each epoch; that for larger regularizer values $\mathbf{w}^T\mathbf{w}$ is smaller; and that the accuracy settles down to about 0.8 very quickly; and that there isn't much difference between the two training regimes. All of these points are relatively typical of stochastic gradient descent with very large datasets.

## 5.4 PRACTICAL METHODS FOR BUILDING CLASSIFIERS

We have described several apparently very different classifiers here. But which classifier should one use for a particular application? Generally, this should be dealt with as a practical rather than a conceptual question: that is, one tries several, and uses the one that works best. With all that said, experience suggests that the first thing to try for most problems is a linear SVM or logistic regression, which tends to be much the same thing. Nearest neighbor strategies are always useful, and are consistently competitive with other approaches when there is lots of training data and one has some idea of appropriate relative scaling of the features. The main difficulty with nearest neighbors is actually finding the nearest neighbors of a query. Approximate methods are now very good, and are reviewed in Section 7.2.1.

The attraction of these methods is that it is relatively easy to build multi-class classifiers, and to add new classes to a system of classifiers.

### 5.4.1 Manipulating Training Data to Improve Performance

Generally, more training data leads to a better classifier. However, training classifiers with large datasets can be difficult, and it can be hard to get enough training data. Typically, only a relatively small number of example items are really important in determining the behavior of a classifier (we see this phenomenon in greater detail in Section **??**). The really important examples tend to be rare cases that are quite hard to discriminate. This is because these cases affect the position of the decision boundary most significantly. We need a large dataset to ensure that these cases are present.

There are some useful tricks that help.

We train on a subset of the examples, run the resulting classifier on the rest of the examples, and then insert the false positives and false negatives into the training set to retrain the classifier. This is because the false positives and false negatives are the cases that give the most information about errors in the configuration of the decision boundaries. We may repeat this several times, and in the final stages, we may use the classifier to seek false positives. For example, we might collect pictures from the Web, classify them, and then look at the positives for errors. This strategy is sometimes called **bootstrapping** (the name is potentially confusing because there is an unrelated statistical procedure known as bootstrapping; nonetheless, we're stuck with it at this point).

There is an extremely important variant of this approach called **hard negative mining**. This applies to situations where we have a moderate supply of positive examples, but an immense number of negative examples. In this case we can't use all the negative examples in training, but we need to search for negative examples that are most likely to improve the classifier's performance. We can do so by selecting a set of negative examples, training with these, and then searching the rest of the negative examples to find ones that generate false positives—these are hard negatives. We can iterate the procedure of training and searching for hard negatives; typically, we expand the pool of negative examples at each iteration.

Most basic classifier training algorithms expect that (a) the percentage of positive and negative examples in the training data set accurately reflects the test data and (b) that the cost of a false positive is the same as the cost of a false negative. On occasion, you will encounter data where one or the other assumption is not true. As an example, if you have one training positive example for every 10, 000 negative training examples, classifying everything as negative is a very good rule *if* the test data reflects those frequencies. But doing so may not be particularly useful, either because the test data doesn't have those frequencies — maybe you had a hard time finding training examples, or were lazy — or because the cost of a false negative is very high. In such cases you can **reweight** the data. For example, you might count each positive example 10 times rather than once when you compute the training loss. Most classifier codes have mechanisms to allow this. Choosing a weighting can be tricky, but it is usual to try several different weightings, then evaluate on a validation set to see which produces the most satisfactory results.

### 5.4.2 Building Multi-Class Classifiers Out of Binary Classifiers

There are two standard methods to build multi-class classifiers out of binary classifiers. In the **all-vs-all** approach, we train a binary classifier for each pair of classes. To classify an example, we present it to each of these classifiers. Each classifier decides which of two classes the example belongs to, then records a vote for that class. The example gets the class label with the most votes. This approach is simple, but scales very badly with the number of classes.

In the **one-vs-all** approach, we build a binary classifier for each class. This classifier must distinguish its class from all the other classes. We then take the class with the largest classifier score. One possible concern with this method is that training algorithms usually do not compel classifiers to be good at ranking examples. We train classifiers so that they give positive scores for positive examples, and negative scores for negative examples, but we do nothing explicit to ensure that a more positive score means the example is more like the positive class. Another important concern is that the classifier scores must be calibrated to one another, so that when one classifier gives a larger positive score than another, we can be sure that the first classifier is more certain than the second. Some classifiers, such as logistic regression, report posterior probabilities, which require no calibration. Others, such as the SVM, report numbers with no obvious semantics and need to be calibrated. The usual method to calibrate these numbers is an algorithm due to **?**, which uses logistic regression to fit a simple probability model to SVM outputs. One-vs-all methods tend to be reliable and effective even when applied to uncalibrated classifier outputs, most likely because training algorithms do tend to encourage classifiers to rank examples correctly.

Neither strategy is particularly attractive when the number of classes is large, because the number of classifiers we must train scales poorly (linearly in one case, quadratically in the other) with the number of classes. If we were to allocate each class a distinct binary vector, we would need only $\log N$ bits in the vector for $N$ classes. We could then train one classifier for each bit, and we should be able to classify into $N$ classes with only $\log N$ classifiers. This strategy tends to founder on questions of which class should get which bit string, because this choice has significant effects on the ease of training the classifiers. Nonetheless, it gives an argument that suggests that we should not need as many as $N$ classifiers to tell $N$ classes apart.

### 5.4.3 Class Confusion Matrices

Evaluating a multi-class classifier is more complex than evaluating a binary classifier. There are only two kinds of mistake that a binary classifier can make. A multi-class classifier can make many more (mapping any one class to any other class). It is useful to know the total error rate of the classifier, which is the percentage of classification attempts that produce the wrong answer. An alternative is the accuracy, which is the percentage of classification attempts that produce the *right* answer. If the error rate is low enough, or the accuracy is high enough, there's not much to worry about. But if it's not, you can look at the **class confusion matrix** to see what's going on.

Table 5.1 gives an example. This is a class confusion matrix from a classifier

| | Predict 0 | Predict 1 | Predict 2 | Predict 3 | Predict 4 | Class error |
|---|---|---|---|---|---|---|
| True 0 | 151 | 7 | 2 | 3 | 1 | 7.9% |
| True 1 | 32 | 5 | 9 | 9 | 0 | 91% |
| True 2 | 10 | 9 | 7 | 9 | 1 | 81% |
| True 3 | 6 | 13 | 9 | 5 | 2 | 86% |
| True 4 | 2 | 3 | 2 | 6 | 0 | 100% |

TABLE 5.1: The class confusion matrix for a multiclass classifier. Further details about the dataset and this example appear in worked example 1.

built on a dataset where one tries to predict the degree of heart disease from a collection of physiological and physical measurements. There are five classes ($0 \dots 4$). The $i$, $j$'th cell of the table shows the number of data points of true class $i$ that were classified to have class $j$. As I find it hard to recall whether rows or columns represent true or predicted classes, I have marked this on the table. For each row, there is a class error rate, which is the percentage of data points of that class that were misclassified. The first thing to look at in a table like this is the diagonal; if the largest values appear there, then the classifier is working well. This clearly isn't what is happening for table 5.1. Instead, you can see that the method is very good at telling whether a data point is in class 0 (the class error rate is rather small), but cannot distinguish between the other classes. This is a strong hint that the data can't be used to draw the distinctions that we want. It might be a lot better to work with a different set of classes.

### 5.4.4 Software for SVM's

We obtain a support vector machine by solving one of the constrained optimization problems given above. These problems have quite special structure, and one would usually use one of the many packages available on the web for SVMs to solve them.

LIBSVM (which can be found using Google, or at `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`) is a dual solver that is now widely used; it searches for nonzero Lagrange multipliers using a clever procedure known as SMO (sequential minimal optimization). A good primal solver is PEGASOS; source code can be found using Google, or at `http://www.cs.huji.ac.il/~shais/code/index.html`.

SVMLight (Google, or `http://svmlight.joachims.org/`) is a comprehensive SVM package with numerous features. It can produce sophisticated estimates of the error rate, learn to rank as well as to classify, and copes with hundreds of thousands of examples. Andrea Vedaldi, Manik Varma, Varun Gulshan, and Andrew Zisserman publish code for a multiple kernel learning-based image classifier at `http://www.robots.ox.ac.uk/~vgg/software/MKL/`. Manik Varma publishes code for general multiple-kernel learning at `http://research.microsoft.com/en-us/um/people/manik/code/GMKL/download.html`, and for multiple-kernel learning using SMO at `http://research.microsoft.com/en-us/um/people/manik/code/SMO-MKL/download.html`. Peter Gehler and Sebastian Nowozin publish code for their recent multiple-kernel learning method at `http://www.vision.ee.ethz.ch/~pgehler/projects/iccv09/index.html`.

CHAPTER 6

# Classifying with Random Forests

I described a classifier as a rule that takes a feature, and produces a class. One way to build such a rule is with a sequence of simple tests, where each test is allowed to use the results of all previous tests. This class of rule can be drawn as a tree (Figure **??**), where each node represents a test, and the edges represent the possible outcomes of the test. To classify a test item with such a tree, you present it to the first node; the outcome of the test determines which node it goes to next; and so on, until the example arrives at a leaf. When it does arrive at a leaf, we label the test item with the most common label in the leaf. This object is known as a **decision tree**.

Figure **??** shows a simple 2D dataset with four classes, next to a decision tree that will correctly classify at least the training data. Actually classifying data with a tree like this is straightforward. We take the data item, and pass it down the tree. Notice it can't go both left and right, because of the way the tests work. This means each data item arrives at a single leaf. We take the most common label at the leaf, and give that to the test item.

The important question is how to get the tree from data. It turns out that the best approach for building a tree incorporates a great deal of randomness. As a result, we will get a different tree each time we train a tree on a dataset. None of the individual trees will be particularly good (they are often referred to as "weak learners"). The natural thing to do is to produce many such trees (a **decision forest**), and allow each to vote; the class that gets the most votes, wins. This strategy is extremely effective.

## 6.1 BUILDING A DECISION TREE

There are many algorithms for building decision trees. We will use an approach chosen for simplicity and effectiveness; be aware there are others. We will always use a binary tree, because it's easier to describe and because that's usual (it doesn't change anything important, though). Each node has a **decision function**, which takes data items and returns either 1 or -1.

We train the tree by thinking about its effect on the training data. We pass the whole pool of training data into the root. Any node splits its incoming data into two pools, left (all the data that the decision function labels 1) and right (ditto, -1). Finally, each leaf contains a pool of data, which it can't split because it is a leaf.

Training the tree uses a straightforward algorithm. First, we choose a class of decision functions to use at each node. It turns out that a very effective algorithm is to choose a single feature at random, then test whether its value is larger than, or smaller than a threshold. For this approach to work, one needs to be quite careful about the choice of threshold, which is what we describe in the next section. Some minor adjustments, described below, are required if the feature chosen isn't ordinal.

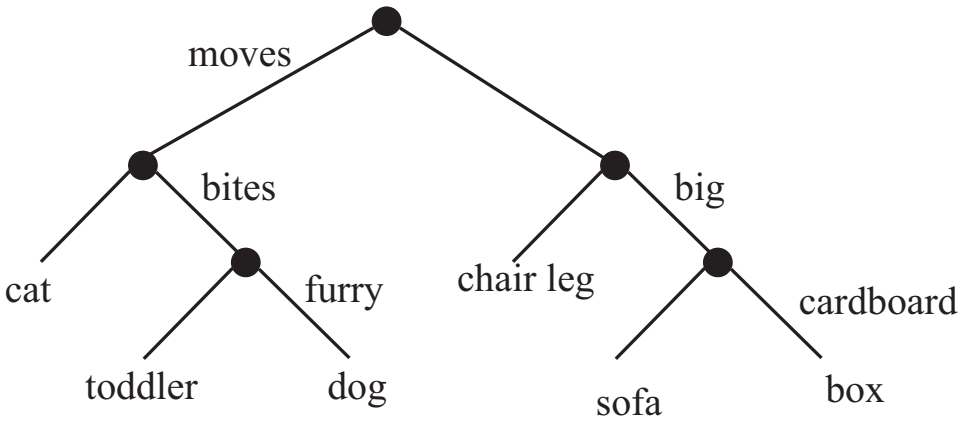


FIGURE 6.1: *This — the household robot's guide to obstacles — is a typical decision tree. I have labelled only one of the outgoing branches, because the other is the negation. So if the obstacle moves, bites, but isn't furry, then it's a toddler. In general, an item is passed down the tree until it hits a leaf. It is then labelled with the leaf's label.*

Surprisingly, being clever about the choice of *feature* doesn't seem add a great deal of value. We won't spend more time on other kinds of decision function, though there are lots.

Now assume we use a decision function as described, and we know how to choose a threshold. We start with the root node, then recursively either split the pool of data at that node, passing the left pool left and the right pool right, or stop splitting and return. Splitting involves choosing a decision function from the class to give the "best" split for a leaf. The main questions are how to choose the best split (next section), and when to stop.

Stopping is relatively straightforward. Quite simple strategies for stopping are very good. It is hard to choose a decision function with very little data, so we must stop splitting when there is too little data at a node. We can tell this is the case by testing the amount of data against a threshold, chosen by experiment. If all the data at a node belongs to a single class, there is no point in splitting. Finally, constructing a tree that is too deep tends to result in generalization problems, so we usually allow no more than a fixed depth $D$ of splits. Choosing the best splitting threshold is more complicated.

### 6.1.1 Entropy and Information Gain

Figure 6.2 shows two possible splits of a pool of training data. These splits are obtained by testing the horizontal feature against a threshold. In one case, the left and the right pools contain about the same fraction of positive ('x') and negative ('o') examples. In the other, the left pool is all positive, and the right pool is mostly negative. Clearly this is the better choice of threshold. But we need some way to score what has happened, so we can tell which threshold is best. Notice that, in

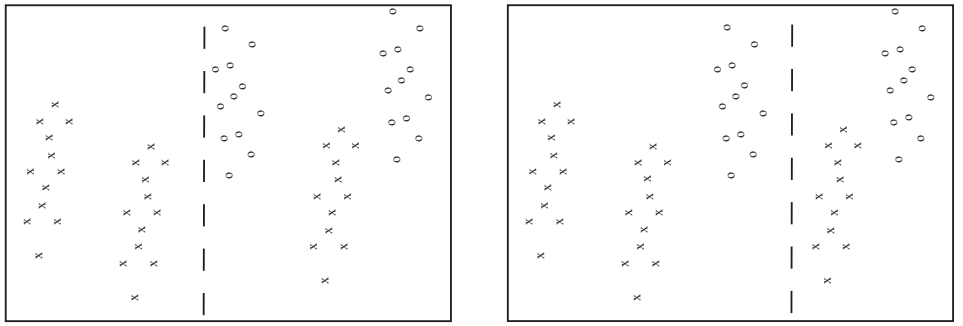


FIGURE 6.2: *Two possible splits of a pool of training data. Positive data is represented with an 'x', negative data with a 'o'. Notice that if we split this pool with the informative line, all the points on the left are 'x's, and two-thirds of the points on the right are 'o's. This means that knowing which side of the split a point lies would give us a good basis for estimating the label. In the less informative case, about two-thirds of the points on the left are 'x's and about half on the right are 'x's — knowing which side of the split a point lies is much less useful in deciding what the label is.*

the uninformative case, knowing that a data item is on the left (or the right) does not tell me much more about the data than I already knew. This is because

$$p(1|\text{left pool}) \approx p(1|\text{parent pool}).$$

In the second case, knowing a data item is on the left classifies it completely. In this case, my uncertainty about what class the data item belongs to is significantly reduced if I know whether it goes left or right. To choose a good threshold, we need to keep track of how informative the split is.

It turns out to be straightforward to keep track of information, in simple cases. We will start with an example. Assume I have 4 classes. There are 8 examples in class 1, 4 in class 2, 2 in class 3, and 2 in class 4. How much information *on average* will you need to send me to tell me the class of a given example? Clearly, this depends on how you communicate the information. You could send me the complete works of Edward Gibbon to communicate class 1; the Encyclopaedia for class 2; and so on. But this would be redundant. The question is how little can you send me. Keeping track of the amount of information is easier if we encode it with bits (i.e. you can send me sequences of '0's and '1's).

Imagine the following scheme. If an example is in class 1, you send me a '1'. If it is in class 2, you send me '01'; if it is in class 3, you send me '001'; and in class 4, you send me '101'. Then the expected number of bits you will send me is

$$p(\text{class} = 1)1 + p(2)2 + p(3)3 + p(4)3 = \frac{1}{2}1 + \frac{1}{4}2 + \frac{1}{8}3 + \frac{1}{8}3$$

which is 1.75 bits. This number doesn't have to be an integer, because it's an expectation.

Notice that for the $i$'th class, you have sent me $-\log_2 p(i)$ bits. We can write the expected number of bits you need to send me as

$$-\sum_i p(i) \log_2 p(i).$$

This expression handles other simple cases correctly, too. You should try what happens if you have two classes, each with 8 examples in them; 256 classes, each with one example in them; and 5 classes, with 16 examples in class 1, 8 in class 2, etc. If you try other examples, you may find it hard to construct a scheme where you can send as few bits *on average* as this expression predicts. It turns out that, in general, the smallest number of bits you will need to send me is given by the expression

$$-\sum_i p(i) \log_2 p(i)$$

under all conditions, though it may be hard or impossible to determine what representation is required to achieve this number.

Now we return to the splits. Write $\mathcal{P}$ for the set of all data at the node. Write $\mathcal{P}_l$ for the left pool, and $\mathcal{P}_r$ for the right pool. The **entropy** of a pool $\mathcal{C}$ is a function $H(\mathcal{C})$ that scores how many bits would be required to represent the class of an item in that pool, on average. Write $n(i; \mathcal{C})$ for the number of items of class $i$ in the pool, and $N(\mathcal{C})$ for the number of items in the pool. Then the entropy of the pool $\mathcal{C}$ is

$$-\sum_i \frac{n(i;\mathcal{C})}{N(\mathcal{C})} \log_2 \frac{n(i;\mathcal{C})}{N(\mathcal{C}}.$$

It is straightforward that $H(\mathcal{P})$ bits are required to classify an item in the parent pool $\mathcal{P}$. For an item in the left pool, we need $H(\mathcal{P}_l)$ bits; for an item in the right pool, we need $H(\mathcal{P}_r)$ bits. If we split the parent pool, we expect to encounter items in the left pool with probability

$$\frac{N(\mathcal{P}_l)}{N(\mathcal{P})}$$

and items in the right pool with probability

$$\frac{N(\mathcal{P}_r)}{N(\mathcal{P})}.$$

This means that, on average, we must supply

$$\frac{N(\mathcal{P}_l)}{N(\mathcal{P})} H(\mathcal{P}_l) + \frac{N(\mathcal{P}_r)}{N(\mathcal{P})} H(\mathcal{P}_r)$$

bits to classify data items if we split the parent pool. Now a good split is one that results in left and right pools that are informative. In turn, we should need fewer bits to classify once we have split than we need before the split. You can see the difference

$$I(\mathcal{P}_l, \mathcal{P}_r; \mathcal{P}) = H(\mathcal{P}) - \left( \frac{N(\mathcal{P}_l)}{N(\mathcal{P})} H(\mathcal{P}_l) + \frac{N(\mathcal{P}_r)}{N(\mathcal{P})} H(\mathcal{P}_r) \right)$$

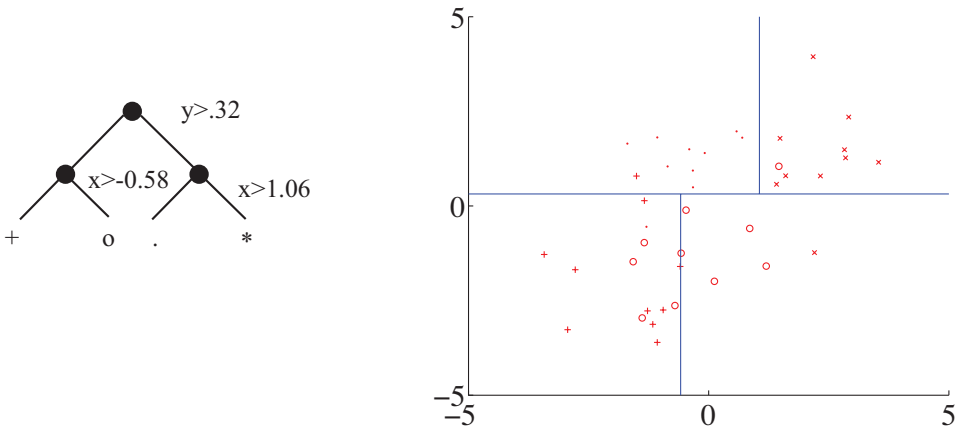


FIGURE 6.3: *A straightforward decision tree*

as the **information gain** caused by the split. This is the average number of bits that you *don't* have to supply if you know which side of the split an example lies. Better splits have larger information gain.

### 6.1.2 Choosing a Split with Information Gain

Recall that our decision function is to choose a feature at random, then test its value against a threshold. Any data point where the value is larger goes to the left pool; where the value is smaller goes to the right. This may sound much too simple to work, but it is actually effective and popular. Assume that we are at a node, which we will label $k$. We have the pool of training examples that have reached that node. The $i$'th example has a feature vector $\mathbf{x}_i$, and each of these feature vectors is a $d$ dimensional vector.

We choose an integer $j$ in the range $1 \ldots d$ uniformly and at random. We will split on this feature, and we store $j$ in the node. Recall we write $x_i^{(j)}$ for the value of the $j$'th component of the $i$'th feature vector. We will choose a threshold $t_k$, and split by testing the sign of $x_i^{(j)} - t_k$. Choosing the value of $t_k$ is easy. Assume there are $N_k$ examples in the pool. Then there are $N_k - 1$ possible values of $t_k$ that lead to different splits. To see this, sort the $N_k$ examples by $x^{(j)}$, then choose values of $t_k$ halfway between example values (Figure **??**). For each of these values, we compute the information gain of the split. We then keep the threshold with the best information gain.

We can elaborate this procedure in a useful way, by choosing $m$ features at random, finding the best split for each, then keeping the feature and threshold value that is best. It is important that $m$ is a lot smaller than the total number of features — a usual root of thumb is that $m$ is about the square root of the total number of features. It is usual to choose a single $m$, and choose that for all the splits.

Now assume we happen to have chosen to work with a feature that isn't

ordinal, and so can't be tested against a threshold. A natural, and effective, strategy is as follows. We can split such a feature into two pools by flipping an unbiased coin for each value — if the coin comes up $H$, any data point with that value goes left, and if it comes up $T$, any data point with that value goes right. We chose this split at random, so it might not be any good. We can come up with a good split by repeating this procedure $F$ times, computing the information gain for each split, then keeping the one that has the best information gain. We choose $F$ in advance, and it usually depends on the number of values the categorical variable can take.

We now have a relatively straightforward blueprint for an algorithm, which I have put in a box. It's a blueprint, because there are a variety of ways in which it can be revised and changed.

**Procedure: 6.1** *Building a decision tree*

Assume we have a data set

**TODO:** an algorithm block

## 6.2 FORESTS

A single decision tree tends to yield poor classifications. One reason is because the tree is not chosen to give the best classification of its training data. We used a random selection of splitting variables at each node, so the tree can't be the "best possible". Obtaining the best possible tree presents significant technical difficulties. It turns out that the tree that gives the best possible results on the training data can perform rather poorly on test data. The training data is a small subset of possible examples, and so must differ from the test data. The best possible tree on the training data might have a large number of small leaves, built using carefully chosen splits. But the choices that are best for training data might not be best for test data.

Rather than build the best possible tree, we have built a tree efficiently, but with number of random choices. If we were to rebuild the tree, we would obtain a different result. This suggests the following extremely effective strategy: build many trees, and classify by merging their results.

### 6.2.1 Building and Evaluating a Decision Forest

There are two important strategies for building and evaluating decision forests. I am not aware of evidence strongly favoring one over the other, but different software packages use different strategies, and you should be aware of the options. In one strategy, we separate labelled data into a training and a test set. We then build multiple decision trees, training each using the whole training set. Finally, we evaluate the forest on the test set. In this approach, the forest has not seen some fraction of the available labelled data, because we used it to test. However, each tree has seen every training data item.

In the other strategy, sometimes called **bagging**, each time we train a tree we

randomly subsample the labelled data with replacement, to yield a training set the same size as the original set of labelled data. Notice that there will be duplicates in this training set, which is like a bootstrap replicate. This training set is often called a **bag**. We keep a record of the examples that do not appear in the bag (the "out of bag" examples). Now to evaluate the forest, we evaluate each tree on its out of bag examples, and average these error terms. In this approach, the entire forest has seen all labelled data, and we also get an estimate of error, but no tree has seen all the training data.

### 6.2.2 Classifying Data Items with a Decision Forest

Once we have a forest, we must classify test data items. There are two major strategies. The simplest is to classify the item with each tree in the forest, then take the class with the most votes. This is effective, but discounts some evidence that might be important. For example, imagine one of the trees in the forest has a leaf with many data items with the same class label; another tree has a leaf with exactly one data item in it. One might not want each leaf to have the same vote.

An alternative strategy that takes this observation into account is to pass the test data item down each tree. When it arrives at a leaf, we record one vote for each of the training data items in that leaf. The vote goes to the class of the training data item. Finally, we take the class with the most votes. This approach allows big, accurate leaves to dominate the voting process. Both strategies are in use, and I am not aware of compelling evidence that one is always better than the other. This may be because the randomness in the training process makes big, accurate leaves uncommon in practice.

Notice one of the major attractions of random forests. Our strategy doesn't depend on the number of classes we are dealing with (though the results might).

**Worked example 6.1** *Classifying heart disease data*

Build a random forest classifier to classify the "heart" dataset from the UC Irvine machine learning repository. The dataset is at `http://archive.ics.uci.edu/ml/datasets/Heart+Disease`. There are several versions. You should look at the processed Cleveland data, which is in the file "processed.cleveland.data.txt".

**Solution:** I used the R random forest package. This uses a bagging strategy. There is sample code in listing **??**. This package makes it quite simple to fit a random forest, as you can see. In this dataset, variable 14 (V14) takes the value 0, 1, 2, 3 or 4 depending on the severity of the narrowing of the arteries. Other variables are physiological and physical measurements pertaining to the patient (read the details on the website). I tried to predict all five levels of variable 14, using the random forest as a multivariate classifier. This works rather poorly, as the out-of-bag class confusion matrix below shows. The total out-of-bag error rate was 45%.

| | Predict 0 | Predict 1 | Predict 2 | Predict 3 | Predict 4 | Class error |
|---|---|---|---|---|---|---|
| True 0 | 151 | 7 | 2 | 3 | 1 | 7.9% |
| True 1 | 32 | 5 | 9 | 9 | 0 | 91% |
| True 2 | 10 | 9 | 7 | 9 | 1 | 81% |
| True 3 | 6 | 13 | 9 | 5 | 2 | 86% |
| True 4 | 2 | 3 | 2 | 6 | 0 | 100% |

This is the example of a class confusion matrix from table 5.1. Fairly clearly, one can predict narrowing or no narrowing from the features, but not the degree of narrowing (at least, not with a random forest). So it is natural to quantize variable 14 to two levels, 0 (meaning no narrowing), and 1 (meaning any narrowing, so the original value could have been 1, 2, or 3). I then built a random forest to predict this from the other variables. The total out-of-bag error rate was 19%, and I obtained the following out-of-bag class confusion matrix

| | Predict 0 | Predict 1 | Class error |
|---|---|---|---|
| True 0 | 138 | 26 | 16% |
| True 1 | 31 | 108 | 22% |

Notice that the false positive rate (16%, from 26/164) is rather better than the false negative rate (22%). Looking at these class confusion matrices, you might wonder whether it is better to predict $0, \ldots, 4$, then quantize. But this is not a particularly good idea. While the false positive rate is 7.9%, the false negative rate is much higher (36%, from 50/139). In this application, a false negative is likely more of a problem than a false positive, so the tradeoff is unattractive.

Listing 6.1: R code used for the random forests of worked example 1

```
setwd('/users/daf/Current/courses/Probcourse/Trees/RCode');
install.packages('randomForest')
library(randomForest)
heart<-read.csv('processed.cleveland.data.txt', header=FALSE)
heart$levels<-as.factor(heart$V14)
heartforest.allvals<-
  randomForest(formula=levels~V1+V2+V3+V4+V5+V6
               +V7+V8+V9+V10+V11+V12+V13,
               data=heart, type='classification', mtry=5)
# this fits to all levels
# I got the CCM by typing
heartforest.allvals
heart$yesno<-cut(heart$V14, c(-Inf, 0.1, Inf))
heartforest<-
  randomForest(formula=yesno~V1+V2+V3+V4+V5+V6
               +V7+V8+V9+V10+V11+V12+V13,
               data=heart, type='classification', mtry=5)
# this fits to the quantized case
# I got the CCM by typing
heartforest
```

CHAPTER 7

# Exploiting your Neighbors

## 7.1 CLASSIFYING WITH NEAREST NEIGHBORS

Example points near an unclassified point should indicate the class of that point. **Nearest neighbors** methods build classifiers using this heuristic. We could classify a point by using the class of the nearest example whose class is known, or use several example points and make them vote. It is reasonable to require that some minimum number of points vote for the class we choose.

A $(k, l)$ nearest neighbor classifier finds the $k$ example points closest to the point being considered, and classifies this point with the class that has the highest number of votes, as long as this class has more than $l$ votes (otherwise, the point is classified as unknown). A $(k, 0)$-nearest neighbor classifier is usually known as a **k-nearest neighbor classifier**, and a $(1, 0)$-nearest neighbor classifier is usually known as a **nearest neighbor classifier**.

Nearest neighbor classifiers are known to be good, in the sense that the risk of using a nearest neighbor classifier with a sufficiently large number of examples lies within quite good bounds of the Bayes risk. As $k$ grows, the difference between the Bayes risk and the risk of using a $k$-nearest neighbor classifier goes down as $1/\sqrt{k}$. In practice, one seldom uses more than three nearest neighbors. Furthermore, if the Bayes risk is zero, the expected risk of using a $k$-nearest neighbor classifier is also zero. Finding the $k$ nearest points for a particular query can be difficult, and Section 7.2.1 reviews this point.

A second difficulty in building such classifiers is the choice of distance. For features that are obviously of the same type, such as lengths, the usual metric may be good enough. But what if one feature is a length, one is a color, and one is an angle? One possibility is to whiten the features (section **??**). This may be hard if the dimension is so large that the covariance matrix is hard to estimate. It is almost always a good idea to scale each feature independently so that the variance of each feature is the same, or at least consistent; this prevents features with very large scales dominating those with very small scales.

Notice that nearest neighbors (fairly obviously) doesn't like categorical data. If you can't give a clear account of how far apart two things are, you shouldn't be doing nearest neighbors. It is possible to fudge this point a little, by (say) putting together a distance between the levels of each factor, but it's probably unwise.

### 7.1.1 Using Nearest Neighbors in Practice

As you'd expect, R has nearest neighbor code that seems quite good (I haven't had any real problems with it, at least). There isn't really all that much to say about the code; I give an example below.

**Worked example 7.1** *Classifying using nearest neighbors*

Build a nearest neighbor classifier to classify the digit data originally constructed by Yann Lecun. You can find it at several places. The original dataset is at `http://yann.lecun.com/exdb/mnist/`. The version I used was used for a Kaggle competition (so I didn't have to decompress Lecun's original format). I found it at `http://www.kaggle.com/c/digit-recognizer`.

**Solution:** I used the R FNN package. This uses a bagging strategy. There is sample code in listing **??**. I trained on 1000 of the 42000 examples, so you could see how in the code. I tested on the next 200 examples. For this (rather small) case, I found the following class confusion matrix

| | P | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 20 | 4 | 1 | 0 | 1 | 0 | 2 | 2 | 1 |
| 2 | 0 | 0 | 20 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 12 | 0 | 0 | 0 | 0 | 4 | 0 |
| 4 | 0 | 0 | 0 | 0 | 18 | 0 | 0 | 0 | 1 | 1 |
| 5 | 0 | 0 | 0 | 0 | 0 | 19 | 0 | 0 | 1 | 0 |
| 6 | 1 | 0 | 0 | 0 | 0 | 0 | 18 | 0 | 0 | 0 |
| 7 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 19 | 0 | 2 |
| 8 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 16 | 0 |
| 9 | 0 | 0 | 0 | 2 | 3 | 1 | 0 | 1 | 1 | 14 |

There are no class error rates here, because I was in a rush and couldn't recall the magic line of R to get them. However, you can see the classifier works rather well for this case.

## 7.2 FINDING YOUR NEAREST NEIGHBORS

### 7.2.1 Finding the Nearest Neighbors and Hashing

To build a nearest neighbors classifier, we need to find the members of a set of high dimensional vectors that are closest to some query vector. It turns out this is a general, and quite difficult, problem. A linear search through the dataset is fine for a small set of data items, but we will operate at scales where we need something more efficient. Surprisingly, exact solutions turn out to be only very slightly more efficient than a linear search through the dataset. Approximate solutions are much more efficient. They are approximate in the sense that, with high probability, they return a point that is almost as close to the query as the closest point. The main trick to obtaining a good approximate solution is to carve the space into cells, then look at items that lie in cells near the query vector; there are two methods that are worth discussing in detail here.

#### Locality Sensitive Hashing

Listing 7.1: R code used for the Nearest neighbors of worked example 1

```
setwd('/users/daf/Current/courses/Probcourse/NearestNeighbors/RCode');
# install.packages(FNN)
library(FNN)

ortrain <- read.csv("KNN-Digits/train.csv", header=TRUE)
# now we need to do a train test split
nrow(ortrain)
#42000 rows
# order appears random
wtrain<-ortrain[1:1000, 2:784]
wtrl<-ortrain[1:1000, 1]
wtest<-ortrain[1001:1200, 2:784]
results <- (0:9)[knn(wtrain, wtest, wtrl,
                     k = 10, algorithm="cover_tree")]
wtel<-ortrain[1001:1200, 1]
newdata<-data.frame(v1=as.factor(as.matrix(results)),
                    v2=as.factor(as.matrix(wtel)))
rccm<-table(newdata)

# this gives me the class confusion matrix
```

In **locality sensitive hashing**, we build a set of hash tables containing the data items, using different hashing functions for each table. For a query item, we recover whatever is in each hash table at the location corresponding to the hash code computed for the query item. We search this set, keeping any data items from this set that are sufficiently close to the query. There are many choices of hash function; the most widely used in vision is **random projection**. Write $\mathbf{v}$ for a vector, representing either a query or a data item. We now obtain a single bit of a hash code by choosing a random vector $\mathbf{r}$ and then computing $sign(\mathbf{v} \cdot \mathbf{r})$. Computing a $k$-bit hash code involves choosing $k$ such random vectors, then computing one bit for each. There is a set of $k$ such random vectors associated with each hash table. Geometrically, choosing an $\mathbf{r}$ corresponds to choosing a hyperplane in the data space, and the hashing bit corresponds to which side of the hyperplane $\mathbf{v}$ lies on. A $k$-bit hash code identifies a cell in an arrangement of $k$ hyperplanes in which $\mathbf{v}$ lies. $k$ will be small compared to the dimension, and so we are cutting the space into $2^k$ cells. This means that there will be relatively few data items that lie in the same cell as a query. Some nearby data items may not lie in the same cell, because they could be on the other side of a hyperplane, but these items should lie in the same cell in another hash table.

All these assertions can be made precise, resulting in a guarantee that: (a) a data item that is almost as close as the nearest neighbor will be found with high probability; and (b) all data items closer to the query than some threshold will be found with high probability, whereas data items that are significantly more distant

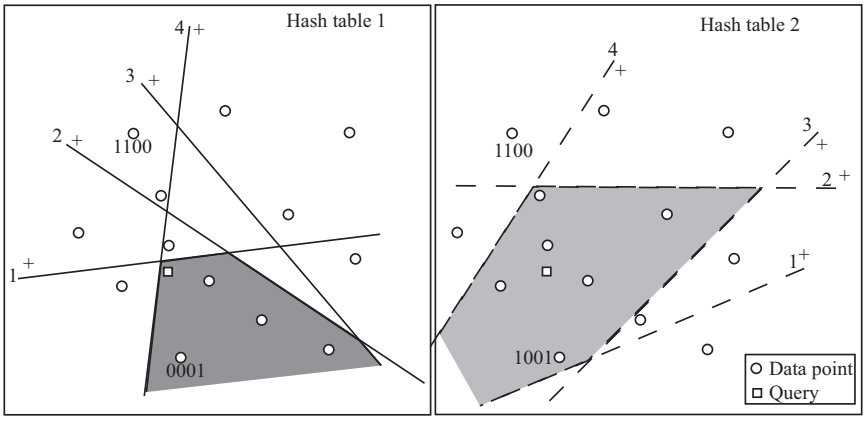


FIGURE 7.1: *In locality sensitive hashing using a random projection hash function, the hash function is equivalent to a hyperplane in the data space. Items that lie on one side of the hyperplane corresponding to the nth bit have that bit set to one; otherwise, it is zero. These hyperplanes cut the space of data into a set of cells. All the data items in a cell get a binary hash code (shown for two points in each figure; we have marked the order of the bits by labeling the hyperplanes, and the +s show which side of the hyperplane gets a one). To query, we find all data items in the same hash table entry as the query (the filled polygons in the figure), and then find the closest. However, the nearest neighbor might not be in this cell (for example, the case on the* **left***). To reduce the probability of error from this cause, we use more than one hash table and search the union of the sets of points lying in the query cell. In the case illustrated, the nearest neighbor of the query lies in the query cell for the second hash table, on the* **right***. The hash tables reduce the set of points we need to search, with high probability of finding a point that is almost as close as the nearest neighbor.*

will be found with low probability. Straightforward geometric intuition suggests that this approach will work best when the data items have zero mean, which is easy to arrange. Notice that using $n$ $k$-bit hash tables is not the same as using one $nk$-bit hash table. In the first case, the list of points returned from a particular query is a union of the lists returned from each of the $n$ hash tables. This means that points that are near the query but just happen to lie outside the query's cell for one hash table, have a good chance of being found in another hash table. In the second case, the list we must handle is much shorter (because there are more cells), but there is a better chance of missing nearby points. The choice of $n$ and $k$ will depend on dimension and on the nature of the probabilistic guarantee one wants. There are a variety of other possible choices of hash function. Details of other choices, and precise statements of the relevant guarantees, can be found in (**?**).

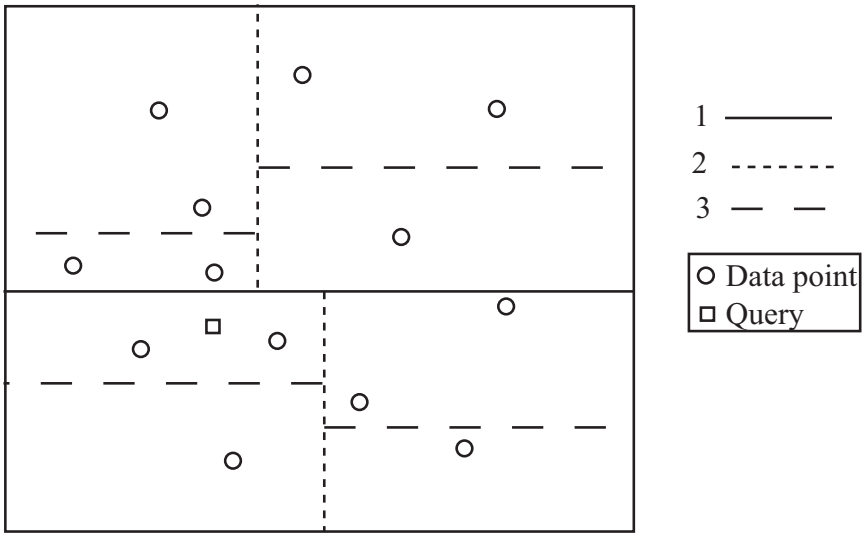


FIGURE 7.2: *A k-d tree is built by recursively splitting cells along dimensions. The order in which cells are split for this tree is shown by the dashes on the lines. The nearest neighbor for the query point is found by (a) finding the closest item in the query point's cell, then (b) backtracking, and looking at cells that could contain closer items. Notice that in this example one will need to go right up to the root of the tree and down the other side to find the nearest neighbor. In high dimensions, this backtracking becomes intractable, but if an approximate nearest neighbor is sufficient, the amount of backtracking can be controlled successfully.*

**KD-Trees for Approximate Nearest Neighbors**

Random projection methods build a cell structure that is independent of the distribution of the data. This means trouble if data is heavily concentrated in some regions, because queries that land in a heavily populated cell of the hash table will need to search a long list. An alternative method is to use a **k-d tree** to build the cell structure. A k-d tree is built by recursively splitting cells. The root will be the whole space. To generate the children of a cell, select one dimension $d$, perhaps at random, and select some threshold value $t_d$. Write the $d$th component of $\mathbf{v}$ as $v_d$. Now all data items in a cell with $v_d \leq t_d$ are placed in the left child, and all others in the right. We now apply this splitting procedure recursively to the root, until the children are sufficiently small. If we choose the threshold value appropriately (for example, the median of the data in the cell), we can ensure that cells are small in dense components of the space and large in sparse components.

The nearest neighbor to a query can then be found by walking the tree to find the cell containing the query point. We then check any data items in that cell. Write the distance from the query to the closest as $d_c$. We now backtrack, investigating cells that could contain points closer than $d_c$ and updating $d_c$ when we find a better point. We can prune any branch of the tree that represents a volume that is further from the query than $d_c$. This procedure works well for low

dimensions, but becomes unattractive in high dimensions because we will need to explore too many cells (the number of neighbors of a cell goes up exponentially with dimension).

This difficulty can be avoided if an approximate nearest neighbor is sufficient. In the **best bin first** approach, we look at a fixed number $N_c$ of cells, then report the best point found so far. Promising cells will tend to have some points that are close to the query, and we define the distance between a cell and the query to be the shortest distance from the query to any point on the cell's boundary. Whenever we investigate the child of a cell, we insert the other child into a priority queue, ordered by distance to the query. Once we have checked a cell, we retrieve the next cell from the priority queue. We do this until we have looked at $N_c$ cells. We will look mainly at cells that are close to the query, and so the point we report is a good approximate nearest neighbor.

Good performance of a particular method depends somewhat on the dataset. For most applications, the choice of method can be made offline using the dataset, or a subset of it. **?** describe a software package that can choose an approximate nearest neighbors method that is fastest for a particular dataset. Generally, they find that using multiple randomized k-d trees is usually the best; at the time of writing, software could be found at `http://www.cs.ubc.ca/~mariusm/index.php/FLANN/FLANN`.

CHAPTER 8

# Regression

Classification tries to predict a class from a data item. Regression tries to predict a value. There are several reasons to do this. First, we might actually need a value. For example, we know the zip code of a house, the square footage of its lot, the number of rooms and the square footage of the house, and we wish to predict its likely sale price. As another example, we know the cost and condition of a trading card for sale, and we wish to predict a likely profit in buying it and then reselling it. As yet another example, we have a picture with some missing pixels – perhaps there was text covering them, and we want to replace it – and we want to fill in the missing values. As a final example, you can think of classification as a special case of regression, where we want to predict either $+1$ or $-1$. Predicting values is very useful, and so there are many examples like this.

Second, we might want to compare trends in data. Doing so could make it clear what is really happening. Here is an example from Efron ("Computer-Intensive methods in statistical regression", B. Efron, SIAM Review, 1988). Table 1 shows some data from medical devices, which sit in the body and release a hormone. The data shows the amount of hormone currently in a device after it has spent some time in service, and the time the device spent in service. The data describes devices from three production lots (A, B, and C). Each device, from each lot, is supposed to have the same behavior. The important question is: Are the lots the same? The amount of hormone changes over time, so we can't just compare the amounts currently in each device. Instead, we need to determine the relationship between time in service and hormone, and see if this relationship is different between batches. We can do so by regressing hormone against time.

Figure 8.1 shows how a regression can help. In this case, we have modelled the amount of hormone in the device as

$$a \times (\text{time in service}) + b$$

for $a$, $b$ chosen to get the best fit (much more on this point later!). This means we can plot each data point on a scatter plot, together with the best fitting line. This plot allows us to ask whether any particular batch behaves differently from the overall model in any interesting way.

However, it is hard to evaluate the distances between data points and the best fitting line by eye. A sensible alternative is to subtract the amount of hormone predicted by the model from the amount that was measured. Doing so yields a **residual** — the difference between a measurement and a prediction. We can then plot those residuals (Figure 8.2). In this case, the plot suggests that lot A is special — all devices from this lot contain less hormone than our model predicts.

| Batch A | | Batch B | | Batch C | |
|---|---|---|---|---|---|
| Amount of Hormone | Time in Service | Amount of Hormone | Time in Service | Amount of Hormone | Time in Service |
| 25.8 | 99 | 16.3 | 376 | 28.8 | 119 |
| 20.5 | 152 | 11.6 | 385 | 22.0 | 188 |
| 14.3 | 293 | 11.8 | 402 | 29.7 | 115 |
| 23.2 | 155 | 32.5 | 29 | 28.9 | 88 |
| 20.6 | 196 | 32.0 | 76 | 32.8 | 58 |
| 31.1 | 53 | 18.0 | 296 | 32.5 | 49 |
| 20.9 | 184 | 24.1 | 151 | 25.4 | 150 |
| 20.9 | 171 | 26.5 | 177 | 31.7 | 107 |
| 30.4 | 52 | 25.8 | 209 | 28.5 | 125 |

TABLE 8.1: *A table showing the amount of hormone remaining and the time in service for devices from lot A, lot B and lot C. The numbering is arbitrary (i.e. there's no relationship between device 3 in lot A and device 3 in lot B). We expect that the amount of hormone goes down as the device spends more time in service, so cannot compare batches just by comparing numbers.*

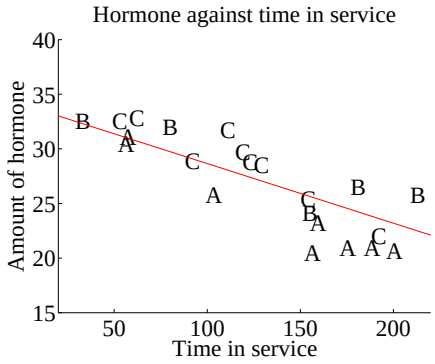


FIGURE 8.1: *A scatter plot of hormone against time for devices from tables 8.1 and 8.1. Notice that there is a pretty clear relationship between time and amount of hormone (the longer the device has been in service the less hormone there is). The issue now is to understand that relationship so that we can tell whether lots A, B and C are the same or different. The best fit line to all the data is shown as well, fitted using the methods of section 8.1.*

## 8.1 LINEAR REGRESSION AND LEAST SQUARES

Assume we have a dataset consisting of a set of $N$ pairs $(\mathbf{x}_i, y_i)$. We think of $y_i$ as the value of some function evaluated at $\mathbf{x}_i$, with some random component added. This means there might be two data items where the $\mathbf{x}_i$ are the same, and the $y_i$ are different. We refer to the $\mathbf{x}_i$ as **explanatory variables** and the $y_i$ is a **dependent variable**. We want to build a model of the dependence between $y$ and $\mathbf{x}$. This model will be used to predict values of $y$ for new values of $\mathbf{x}$, or

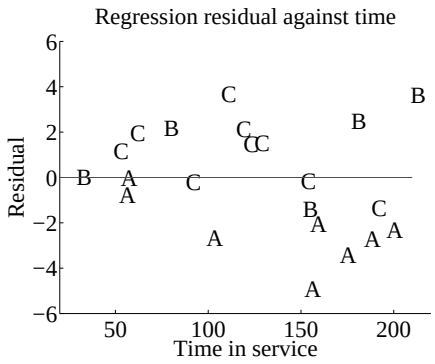


FIGURE 8.2: *This is a scatter plot of residual — the distance between each data point and the best fit line — against time for the devices from tables 8.1 and 8.1. Now you should notice a clear difference; some devices from lots B and C have positive and some negative residuals, but all lot A devices have negative residuals. This means that, when we account for loss of hormone over time, lot A devices still have less hormone in them. This is pretty good evidence that there is a problem with this lot.*

to understand the relationships between the $\mathbf{x}$. The model needs to have some probabilistic component; we do not expect that $y$ is a function of $\mathbf{x}$, and there is likely some error in evaluating $y$ anyhow.

### 8.1.1 Linear Regression

A good, simple model is to assume that the dependent variable is obtained by evaluating a linear function of the explanatory variables, then adding a zero-mean normal random variable. We can write this model as

$$y = \mathbf{x}^T\beta + \xi$$

where $\xi$ is a zero mean normal random variable with unknown variance (we will be able to estimate this later). In this expression, $\beta$ is a vector of weights, which we must estimate. When we use this model to predict a value of $y$ for a particular set of explanatory variables $\mathbf{x}^*$, we cannot predict the value that $\xi$ will take. Our best available prediction is the mean value (which is zero).

**Example: 8.1** *A linear model fitted to a single explanatory variable*

Assume we fit a linear model to a single explanatory variable. Then the model has the form $y = x\beta + \xi$, where $\xi$ is a zero mean random variable. For any value $x^*$ of the explanatory variable, our best estimate of $y$ is $\beta x^*$. In particular, if $x^* = 0$, the model predicts $y = 0$, which is unfortunate. We can draw the model by drawing a line through the origin with slope $\beta$ in the $x$, $y$ plane. The $y$-intercept of this line must be zero.

**Example: 8.2** *A linear model with a non-zero y-intercept*

Assume we have a single explanatory variable, which we write $u$. We can then create a *vector* $\mathbf{x} = [u, 1]^T$ from the explanatory variable. We now fit a linear model to this vector. Then the model has the form $y = \mathbf{x}^T\beta + \xi$, where $\xi$ is a zero mean random variable. For any value $\mathbf{x}^* = [u^*, 1]^T$ of the explanatory variable, our best estimate of $y$ is $(\mathbf{x}^*)^T\beta$, which can be written as $y = \beta_1 u^* + \beta_2$. If $x^* = 0$, the model predicts $y = \beta_2$. We can draw the model by drawing a line through the origin with slope $\beta_1$ and $y$-intercept $\beta_2$ in the $x$, $y$ plane.

The next step is to determine $\beta$. Because we have that $P(y|x, \beta)$ is normal, with mean $\mathbf{x}^T\beta$, we can write out the log-likelihood of the data. Write $\sigma^2$ for the variance of $\xi$, which we don't know, but will not need to worry about right now. We have that

$$\begin{aligned}\log \mathcal{L}(\beta) &= \sum_i \log P(y_i|\mathbf{x}_i, \beta) \\ &= \frac{1}{2\sigma^2}\sum_i (y_i - \mathbf{x}_i^T\beta)^2 + \text{term not depending on } \beta\end{aligned}$$

Maximizing the log-likelihood of the data is equivalent to minimizing the negative log-likelihood of the data. Furthermore, the term $\frac{1}{2\sigma^2}$ does not affect the location of the minimum. We must have that $\beta$ minimizes

$$\sum_i (y_i - \mathbf{x}_i^T\beta)^2.$$

We can write all this more conveniently using vectors and matrices. Write $\mathbf{y}$ for the vector

$$\begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix}$$

and $\mathcal{X}$ for the matrix

$$\begin{pmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \ldots \mathbf{x}_n^T \end{pmatrix}.$$

Then we want to minimize

$$(\mathbf{y} - \mathcal{X}\beta)^T(\mathbf{y} - \mathcal{X}\beta)$$

which means that we must have

$$\mathcal{X}^T\mathcal{X}\beta - \mathcal{X}^T\mathbf{y} = 0.$$

For reasonable choices of features, we could expect that $\mathcal{X}^T\mathcal{X}$ — which should strike you as being a lot like a covariance matrix — has full rank. If it does, this equation is easy to solve. If it does not, there is more to do, which we will do below once we have done some examples.

**Procedure: 8.1** *Linear Regression with a Normal Model*

We have a dataset containing $N$ pairs $(\mathbf{x}_i, y_i)$. Each $x_i$ is a $d$-dimensional explanatory vector, and each $y_i$ is a single dependent variable. We assume that each data point conforms to the model

$$y_i = \mathbf{x}_i^T \beta + \xi_i$$

where $\xi_i$ is a normal random variable with mean 0 and unknown variance $\sigma^2$. Write $\mathbf{y}$ for the vector

$$\begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix}$$

and $\mathcal{X}$ for the matrix

$$\begin{pmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \dots \mathbf{x}_n^T \end{pmatrix}.$$

We estimate $\hat{\beta}$ (the value of $\beta$) by solving the linear system

$$\mathcal{X}^T \mathcal{X} \hat{beta} - \mathcal{X}^T \mathbf{y} = 0.$$

The residuals are

$$\mathbf{e} = \mathbf{y} - \mathcal{X}\hat{\beta}.$$

We have that $\mathbf{e}^T \mathbf{1} = 0$. The **mean square error** is given by

$$m = \frac{\mathbf{e}^T \mathbf{e}}{N}.$$

We can estimate $\sigma^2$ by

$$\sigma^2 = \frac{\sum_i (y_i - \mathbf{x}_i^T \hat{\beta})^2}{N} = \frac{\mathbf{e}^T \mathbf{e}}{N} = m.$$

Listing 8.1: R code used for the linear regression example of worked example 1

```
setwd('/users/daf/Current/courses/Probcourse/Regression/RCode/EfronDevices');
efd<-read.table('efrontable.txt',header=TRUE)
summary(efd)
# the table has the form
#N1   Ah   Bh   Ch N2   At   Bt   Ct
# now we need to construct a new dataset
#
hor<-stack(efd, select=2:4)
tim<-stack(efd, select=6:8)
foo<-data.frame(time=tim[, c("values")], hormone=hor[, c("values")])
foo.lm<-lm(hormone~time,data=foo)
summary(foo.lm)
#
```

**Worked example 8.1** *Simple Linear Regression with R*

Regress the hormone data against time for all the devices in the Efron example.

**Solution:** This example is mainly used to demonstrate how to regress in R. There is sample code in listing 8.2. The summary in the listing produces a great deal of information (try it). You will see information about the residuals; the smallest is -4.9357, the largest is 3.7323, and the median, first and third quartiles are there too. You will see the intercept (34.167528) and the coefficient of time (-0.057446), as well as a body of statistical information about these coefficients. The R-squared (which we discuss below) is 0.8688. You can get a figure by doing `plot(foo.lm)`, but these figures will not mean anything yet. We discuss some of them below.

### 8.1.2 Checking Goodness of Fit Qualitatively

It is quite important to know whether the regression is helpful. For example, it seems highly unlikely that regressing the first digit of a telephone number against some numerological score of your name will work that well (or at all). For a dataset with one explanatory variable and one dependent variable, it is quite natural to plot the data on a scatter plot, then plot the model as a line on that scatterplot. Just looking at the picture can be informative (Figure 8.4). But we need more powerful tools, so we can deal with models that are harder to plot.

Assume we have fitted a model $y = \mathbf{x}^T\beta + \xi$ to a dataset, and found the best vector of parameters is $\hat{\beta}$. Then the residual vector is $\mathbf{e} = \mathbf{y} - \mathcal{X}\hat{\beta}$. Inspecting this residual vector will reveal a great deal about a model. We look at qualitative properties first. If the line is horizontal, or close, then the value of the explanatory

variable makes very little contribution to the prediction. This suggests that there is no particular relationship between the two. The prediction will be good only if it should take a constant value. Similarly, if the data points lie far from the regression line, which means that the residual is persistently large compared to the predicted value, then the regression will predict poorly. Figure 8.3 shows a linear regression of age against weight — i.e., this builds a model that tries to predict someone's age from a knowledge of their weight. You won't be surprised that the predictions are poor, which can be confirmed by looking at the figure.

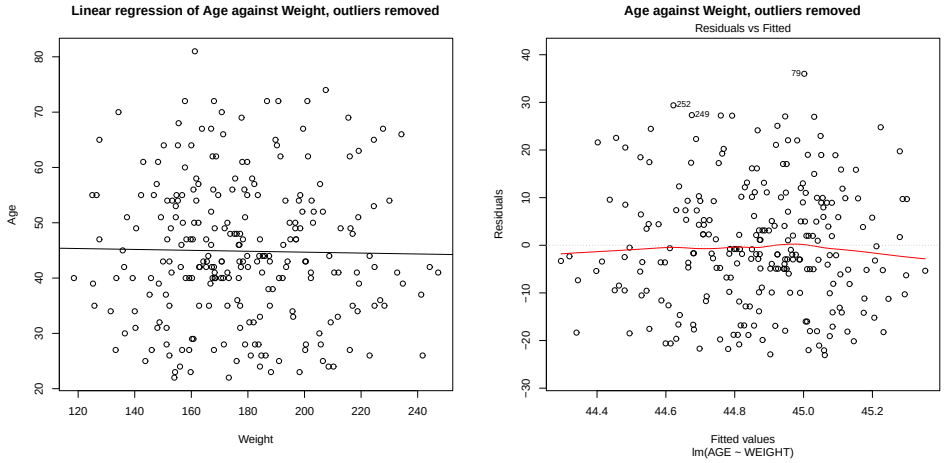


FIGURE 8.3: *On the* **left**, *age regressed against weight for the bodyfat dataset, with four outliers removed. The data are presented as a scatter plot, and the line is the regression line. Notice that the line is nearly horizontal, suggesting that knowing weight does not improve one's prediction of age. Notice also that many data points lie rather far from the line. On the* **right**, *a scatter plot of the residual against the value predicted by the regression. Generally, the residual takes rather large values, suggesting that predictions are unreliable.*

We assumed that $y - \mathbf{x}^T\beta$ was a zero-mean normal random variable with fixed variance. In turn, this means that the value of the residual vector should not depend on the corresponding $y$-value. If it does, this is good evidence that there is a problem. Looking at a scatter plot of $\mathbf{e}$ against $\mathbf{y}$ will often reveal trouble in a regression (Figure 8.4).

In the case of Figure 8.4, the trouble is caused by a few outlying data points severely affecting the regression. We will discuss how to identify and deal with such points in Section 1. Once they have been removed, the regression improves markedly (Figure 8.5).

Figure **??** shows another example, based on the idea of word frequencies. Some words are used very often in text; most are used seldom. The dataset for this figure consists of counts of the number of time a word occurred for the 100 most common words in Shakespeare's printed works. It was originally collected from a concordance, and has been used to attack a variety of interesting questions,


FIGURE 8.4: *On the* **left**, *weight regressed against height for the bodyfat dataset. The data are presented as a scatter plot, and the line is the regression line. Notice the line doesn't describe the data particularly well, because it has been strongly affected by a few data points. On the* **right**, *a scatter plot of the residual against the value predicted by the regression. Large values at some points are another sign of trouble in this regression.*

including an attempt to assess how many words Shakespeare knew. This is hard, because he likely knew many words that he didn't use in his works, so one can't just count. If you look at the plot of Figure 8.11, you can see that a linear regression of count (the number of times a word is used) against rank (how common a word is, 1-100) is not really useful (Figure 8.11). It doesn't model the very high frequencies with which common words are used. You can see this effect in the scatter plot of residual against dependent variable in Figure 8.11 — the residual depends rather strongly on the dependent variable.

### 8.1.3 Evaluating Goodness of Fit

There is an important quantitative measure of how good a regression is. The dependent variable has some variance (unless it's a constant, which makes prediction easy). If our model is of any use, it should explain some aspects of the value of the dependent variable. This means that the variance of the residual should be smaller than the variance of the dependent variable. If the model made perfect predictions, then the variance of the residual should be zero.

We can formalize all this in a relatively straightforward way. We will ensure that $\mathcal{X}$ always has a column of ones in it, so that the regression can have a non-zero y-intercept. We now fit a model

$$\mathbf{y} = \mathcal{X}\beta + \mathbf{e}$$

(where $\mathbf{e}$ is the vector of residual values) by choosing $\beta$ such that $\mathbf{e}^T\mathbf{e}$ is minimized. Then we get some useful technical results.


FIGURE 8.5: *On the* **left**, *weight regressed against height for the bodyfat dataset. I have now removed four data points, which were the most likely to have been outliers. The data are presented as a scatter plot, and the line is the regression line. The line describes the data rather well in comparison with figure 8.4 (notice the axes are different). On the* **right**, *a scatter plot of the residual against the value predicted by the regression. Notice that the mean squared residual is about the same for different values of the prediction; this is consistent with our model, and suggests the regression will yield good predictions.*

**Useful Facts: 8.1** *Regression*

We write $\mathbf{y} = \mathcal{X}\beta + \mathbf{e}$, where $\mathbf{e}$ is the residual. Assume $\mathcal{X}$ has a column of ones, and $\beta$ is chosen to minimize $\mathbf{e}^T\mathbf{e}$. Then we have

1. $\mathbf{e}^T\mathcal{X} = \mathbf{0}$, i.e. that $\mathbf{e}$ is orthogonal to any column of $\mathcal{X}$. This is because, if $\mathbf{e}$ is not orthogonal to some column of $\mathbf{e}$, we can increase or decrease the $\beta$ term corresponding to that column to make the error smaller. Another way to see this is to notice that *beta* is chosen to minimize $\mathbf{e}^T\mathbf{e}$, which is $(\mathbf{y}-\mathcal{X}\beta)^T(\mathbf{y}-\mathcal{X}\beta)$. Now because this is a minimum, the gradient with respect to $\beta$ is zero, so $(\mathbf{y} - \mathcal{X}\beta)^T(-\mathcal{X}) = -\mathbf{e}^T\mathcal{X} = 0$.
2. $\mathbf{e}^T\mathbf{1} = 0$ (recall that $\mathcal{X}$ has a column of all ones, and apply the previous result).
3. $\mathbf{1}^T(\mathbf{y} - \mathcal{X}\beta) = 0$ (same as previous result).
4. $\mathbf{e}^T\mathcal{X}\beta = 0$ (first result means that this is true).

Now $\mathbf{y}$ is a one dimensional dataset arranged into a vector, so we can compute

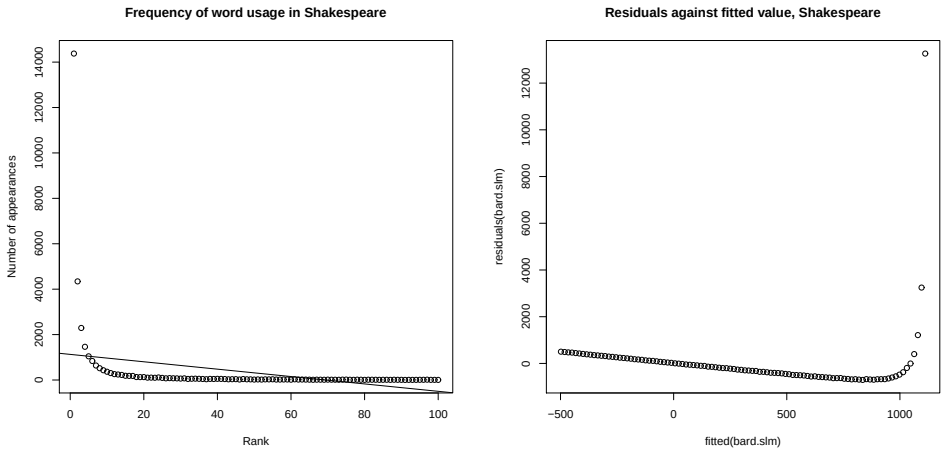


FIGURE 8.6: *On the* **left***, word count plotted against rank for the 100 most common words in Shakespeare, using the data of ****. I show a regression line too. This is a poor fit by eye, confirmed by the plot of the residuals on the* **right***, which shows the residuals plotted against dependent variable for this regression. Notice that the residual value depends very strongly on the value of the dependent variable.*

$\mathsf{mean}\,(\{y\})$ and $\mathsf{var}[y]$. Similarly, $\mathcal{X}\beta$ is a one dimensional dataset arranged into a vector (its elements are $\mathbf{x}_i^T\beta$), as is $\mathbf{e}$, so we know the meaning of mean and variance for each. We have a particularly important result:

$$\mathsf{var}[y] = \mathsf{var}[\mathcal{X}\beta] + \mathsf{var}[e].$$

This is quite easy to show, with a little more notation. Write $\overline{\mathbf{y}} = (1/N)(\mathbf{1}^T\mathbf{y})\mathbf{1}$ for the vector whose entries are all $\mathsf{mean}\,(\{y\})$; similarly for $\overline{e}$ and for $\overline{\mathcal{X}\beta}$. We have

$$\mathsf{var}[y] = (1/N)(\mathbf{y} - \overline{\mathbf{y}})^T(\mathbf{y} - \overline{\mathbf{y}})$$

and so on for $\mathsf{var}[e_i]$, etc. Notice from the facts that $\overline{\mathbf{y}} = \overline{\mathcal{X}\beta}$. Now

$$\begin{aligned}
\mathsf{var}[y] &= (1/N)\left(\left[\mathcal{X}\beta - \overline{\mathcal{X}\beta}\right] + \left[\mathbf{e} - \overline{\mathbf{e}}\right]\right)^T\left(\left[\mathcal{X}\beta - \overline{\mathcal{X}\beta}\right] + \left[\mathbf{e} - \overline{\mathbf{e}}\right]\right) \\
&= (1/N)\left(\left[\mathcal{X}\beta - \overline{\mathcal{X}\beta}\right]^T\left[\mathcal{X}\beta - \overline{\mathcal{X}\beta}\right] + 2\left[\mathbf{e} - \overline{\mathbf{e}}\right]^T\left[\mathcal{X}\beta - \overline{\mathcal{X}\beta}\right] + \left[\mathbf{e} - \overline{\mathbf{e}}\right]^T\left[\mathbf{e} - \overline{\mathbf{e}}\right]\right) \\
&= (1/N)\left(\left[\mathcal{X}\beta - \overline{\mathcal{X}\beta}\right]^T\left[\mathcal{X}\beta - \overline{\mathcal{X}\beta}\right] + \left[\mathbf{e} - \overline{\mathbf{e}}\right]^T\left[\mathbf{e} - \overline{\mathbf{e}}\right]\right) \\
&\quad \text{because } \overline{\mathbf{e}} = 0 \text{ and } \mathbf{e}^T\mathcal{X}\beta = 0 \text{ and } \mathbf{e}^T\mathbf{1} = 0 \\
&= \mathsf{var}[\mathcal{X}\beta] + \mathsf{var}[e].
\end{aligned}$$

This is extremely important, because us allows us to think about a regression as explaining variance in $\mathbf{y}$. As we are better at explaining $\mathbf{y}$, $\mathsf{var}[e]$ goes down. In turn, a natural measure of the goodness of a regression is what percentage of the variance of $\mathbf{y}$ it explains. This is known as $R^2$ (the r-squared measure). We have

$$R^2 = \frac{\mathsf{var}\left[\mathbf{x}_i^T\beta\right]}{\mathsf{var}[y_i]}$$

which gives some sense of how well the regression explains the training data.

Good predictions result in high values of $R^2$, and a perfect model will have $R^2 = 1$ (which doesn't usually happen). For example, the regression of figure 8.1 has an $R^2$ value of 0.87; the regression of weight against all explanatory variables (Figure 8.8) has an $R^2$ value of 0.99; and the regression of Boston house prices against all explanatory variables (Figure 8.8) has an $R^2$ value of 0.72. Similarly, poor predictions result in low values of $R^2$. For example, the regression of weight on height of Figure 8.4 has an $R^2$ of 0.095; the regression of age on weight of Figure 8.3 has an $R^2$ of 0.00030 (because knowing weight tells one essentially nothing about age or height). Removing the outliers from the bodyfat dataset, to get the regression of weight on height of Figure 8.5, improves the $R^2$ to 0.29.

### 8.1.4 Linear Regression: Examples

For the data of tables 8.1 and 8.1, $y$ is the hormone remaining in the device and $\mathbf{x} = (\text{time}, 1)$. This gives us a model of the hormone in the device as

$$y = \beta_1 \text{time} + \beta_2$$

which should look like the line in figure **??**. We get $\beta = (-0.0574, 34.2)$. Now we can ask whether some lots of device behave differently than others. One way to address this is to consider the **residual**,

$$y_i - \mathbf{x}_i^T \beta$$

which is the difference between the observed value and what the model predicts. Look at figure 8.2, which shows this residual plotted against the time. Notice that, for batch A, the model always over predicts, whereas batches B and C seem about the same; this suggests that there is something different about A — any effects caused by the hormone being taken up from the device have been accounted for by our model, and the residual shows the effects that are left.

## 8.2 PRODUCING GOOD LINEAR REGRESSIONS

Outlying data points can significantly weaken the usefulness of a regression. We need to identify data points that might be a problem, and then resolve how to deal with them. One possibility is that they are true outliers — someone recorded a data item wrong, or they represent an effect that just doesn't occur all that often. Another is that they are important data, and our linear model may not be good enough. If the data points really are outliers, we can ignore them; if they aren't, we may be able to improve the regression by transforming features or by finding a new explanatory variable.

### 8.2.1 Problem Data Points

We are solving for the $\beta$ that minimizes $\sum_i (y_i - \mathbf{x}_i^T \beta)^2$, equivalently for the $\beta$ that produces the smallest value of $\sum_i e_i^2$. This means that residuals with large value can have a very strong influence on the outcome — we are squaring that large value. Generally, many residuals of medium size will have a smaller cost that one large residual and the rest tiny. In turn, some data points can have an excessive

influence on the choice of $\beta$. This creates a problem, because data points that are clearly wrong (sometimes called **outliers**) can also have the highest influence on the outcome of the regression. Compare Figure 8.4 and Figure 8.5 to see this effect for a simple case.

When we have only one explanatory variable, there's an easy method to spot problem data points. One produces a scatter plot and a regression line, and the difficulty is usually obvious. In particularly tricky cases, printing the plot and using a perspex ruler to draw a line by eye can help. When there are more, we need more powerful visualization tools.

There are two tools that are simple and effective. One method deletes the $i$'th point, computes the regression for the reduced data set, then compares the true value of *every other point* to the predictions made by the dataset with the $i$'th point deleted. The score for the comparison is called **Cook's distance**. If a point has a large value of Cook's distance, then it has a strong influence on the regression and might well be an outlier. Typically, one computes Cook's distance for each point, and takes a closer look at any point with a large value. This procedure is described in more detail in procedure 2

**Procedure: 8.2** *Computing Cook's distance*

We have a dataset containing $N$ pairs $(\mathbf{x}_i, y_i)$. Each $x_i$ is a $d$-dimensional explanatory vector, and each $y_i$ is a single dependent variable. Write $\hat{\beta}$ for the coefficients of a linear regression (see procedure 1), and $\hat{\beta}_{\hat{i}}$ for the coefficients of the linear regression computed by omitting the $i$'th data point, and $m$ for the mean square error. The Cook's distance of the $i$'th data point is

$$\frac{\sum_j (\mathbf{x}_j^T \hat{\beta} - \mathbf{x}_j^T \hat{\beta}_{\hat{i}})}{dm}.$$

An alternative method identifies data points that have few nearby points by computing **leverage**. Write $\hat{\beta}$ for the estimated value of $\beta$, and $\mathbf{y}_p = \mathcal{X}\hat{\beta}$ for the predicted $y$ values. Then we have

$$\hat{\beta} = \left(\mathcal{X}^T\mathcal{X}\right)^{-1}\left(\mathcal{X}^T\mathbf{y}\right)$$

so that

$$\mathbf{y}_p = \left(\mathcal{X}\left(\mathcal{X}^T\mathcal{X}\right)^{-1}\mathcal{X}^T\right)\mathbf{y}.$$

The matrix $\left(\mathcal{X}\left(\mathcal{X}^T\mathcal{X}\right)^{-1}\mathcal{X}^T\right)$ is sometimes called the **hat matrix**, and is written $\mathcal{H}$. It gives the predicted values as a linear function of the observed values. The leverage of the $i$'th point is the $i$'th diagonal element, $h_{ii}$, of the hat matrix $\mathcal{H}$. It is straightforward to show that the eigenvalues of a hat matrix can be only 1 or 0, and that it is symmetric (exercises). It is also straightforward to show that, for the

$i$'th row of the hat matrix, the sum of squares is less than one, that is

$$\sum_j h_{ij}^2 \leq 1.$$

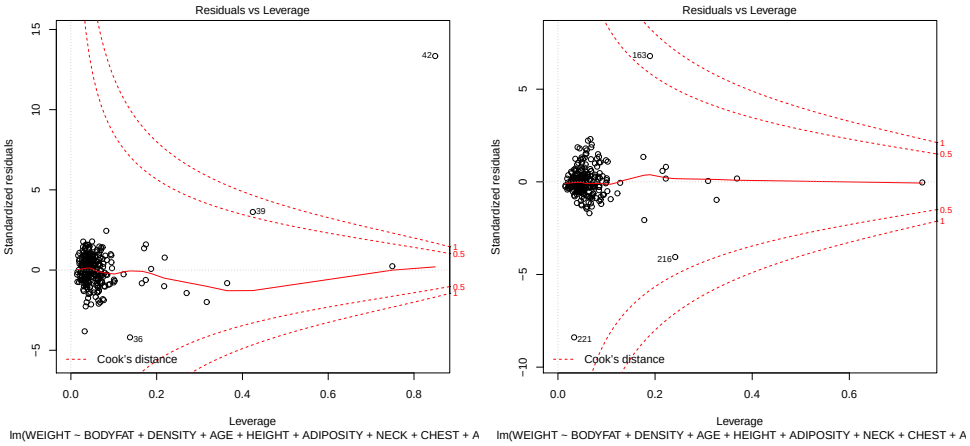


FIGURE 8.7: *On the* **left**, *residuals plotted against leverage for a regression of weight against all other measurements for the bodyfat dataset. I did* not *remove the outliers. The contours on the plot are contours of Cook distance; I have overlaid arrows showing points with suspiciously large Cook distance. Notice also that several points have high leverage, without having a large residual value. These points may or may not present problems. On the* **right**, *the same plot for this dataset with points 36, 39, 41 and 42 removed (these are the points I have been removing for each such plot). Notice that another point now has high Cook distance, but mostly the residual is much smaller.*

You can interpret each row of the hat matrix as a set of mixing weights that mix the observations to produce the prediction for a particular data point. But these weights have limited size, so that if $h_{ii}$ is large, then the other terms must be small. In particular, if $h_{ii}$ is large, then other data items make relatively little contribution to the prediction *at that point*. This could mean that the point is an outlier, or that there are few examples nearby. In either case, it is reasonable to worry about the data point. One might try and get more data, or reduce the number of explanatory variables.

### 8.2.2 Explanatory variables

When there is more than one explanatory variable, it is difficult to plot the regression. Instead, one can get an idea of the usefulness of the regression by plotting the residual against the predicted value. Figure 8.9 show these plots for the bodyfat dataset, and for a regression of boston house prices against a variety of explanatory variables (the data set is quite well known; you can find it at the

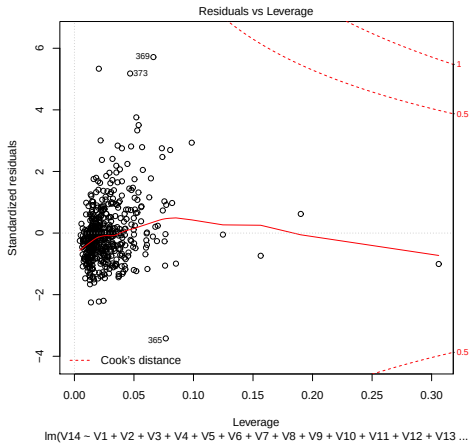


FIGURE 8.8: *Residuals plotted against leverage for a regression of weight against all other measurements for the Boston housing. The contours on the plot are contours of Cook distance; No points have suspiciously large Cook distance, though the software package has identified points most likely to present problems. Notice also that several points have high leverage, without having a large residual value. These points may or may not present problems. The residual plot of Figure 8.9 suggests that this regression will make poor predictions, but the problem does not seem to be outlying points.*

UCI repository, `http://archive.ics.uci.edu/ml/datasets/Housing`). In these plots, you should notice there is some structure — the residuals seem to depend on the predicted value. This suggests that we could improve the regression either by supplying another explanatory variable, or by transforming the variables we have. Both methods are useful.

Sometimes the data isn't in a form that leads to a good linear regression. In this case, transforming explanatory variables, the dependent variable, or both can lead to big improvements. Figure **??** shows one example, based on the idea of word frequencies. Some words are used very often in text; most are used seldom. The dataset for this figure consists of counts of the number of time a word occurred for the 100 most common words in Shakespeare's printed works. It was originally collected from a concordance, and has been used to attack a variety of interesting questions, including an attempt to assess how many words Shakespeare knew. This is hard, because he likely knew many words that he didn't use in his works, so one can't just count.

Notice that a linear regression of count (the number of times a word is used) against rank (how common a word is, 1-100) is not really useful. It doesn't model the very high frequencies with which common words are used. However, if we regress log-count against log-rank, we get a very good fit indeed. This suggests that Shakespeare's word usage (at least for the 100 most common words) is consistent with **Zipf's law**. This gives the relation between frequency $f$ and rank $r$ for a

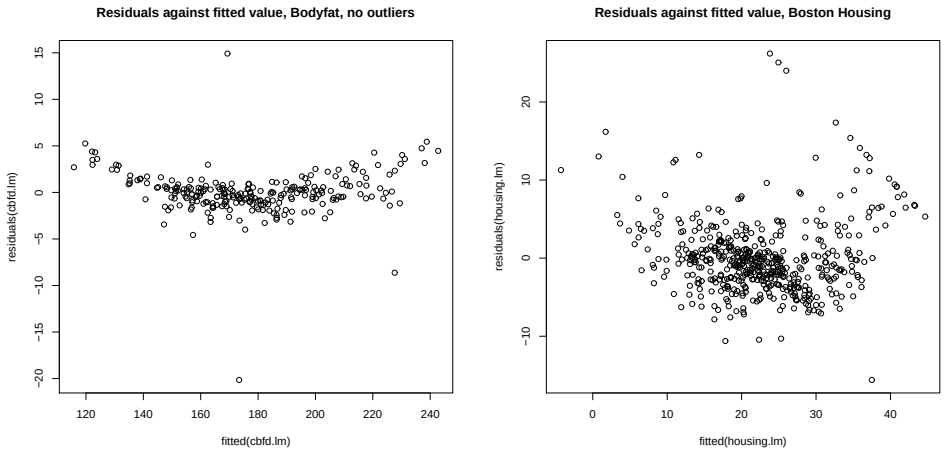


FIGURE 8.9: *On the* **left**, *residuals plotted against predicted value for a regression of weight against all other measurements for the bodyfat dataset, with four outliers removed. This regression is quite successful (small residuals). Notice that the variance of the residuals changes somewhat as the predicted value increases, giving a banana shape to the data. This suggests that we are missing an explanatory variable, or that a non-linear transformation of the explanatory variables might be helpful. On the* **right**, *a scatter plot of the residual against the value predicted by the regression for the price of a house in Boston regressed against a variety of explanatory variables. Generally, the residual takes rather large values. At small values of the prediction, the residual is somewhat larger. This suggests that we are missing an explanatory variable, or that a non-linear transformation of the explanatory variables might be helpful. There is a curious linear structure in the top right corner, which might have been caused by the dependent variable being thresholded.*

word as

$$f \propto \frac{1}{r}^s$$

where $s$ is a constant characterizing the distribution. Our linear regression suggests that $s$ is approximately 1.67 for this data.

In some cases, the natural logic of the problem will suggest variable transformations that improve regression performance. For example, one could argue that humans have approximately the same density, and so that weight should scale as the cube of height; in turn, this suggests that one regress weight against the cube root of height. Generally, shorter people tend not to be scaled versions of taller people, so the cube root might be too aggressive. For example, the body mass index divides weight by the square (rather than the cube) of height. An appropriate transformation has attracted a great deal of interest over the last century and a half, perhaps because there are many confounding variables. For example, the body mass index tends to regard muscular individuals as obese.

In other cases, one might need to try different transformations or groupings of

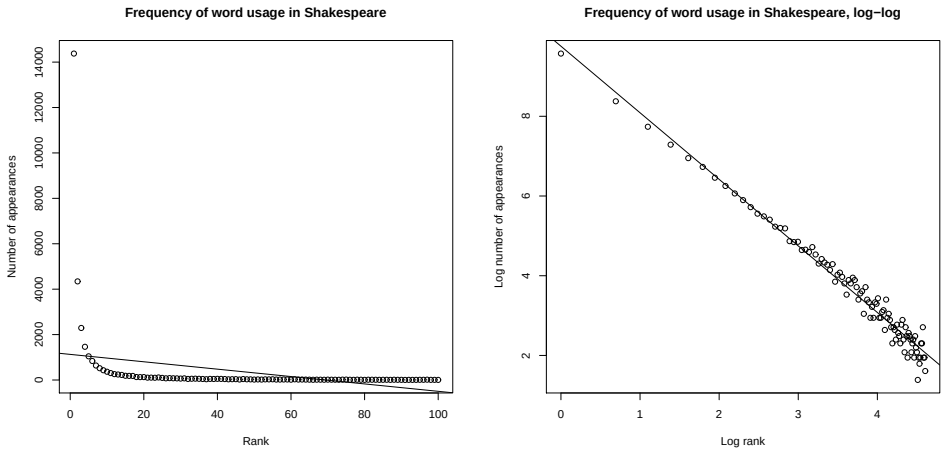


FIGURE 8.10: *On the* **left**, *word count plotted against rank for the 100 most common words in Shakespeare, using the data of ****. I show a regression line too. This is a poor fit by eye, confirmed by Figure* **??**. *On the* **right**, *log word count plotted against log rank for the 100 most common words in Shakespeare, using the data of ****. The regression line is very close to the data, confirmed by Figure* **??**.

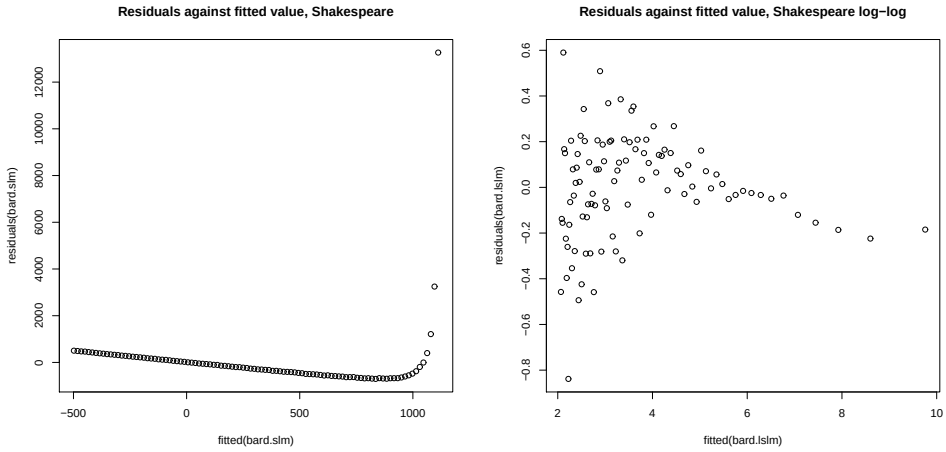


FIGURE 8.11: *On the* **left**, *residuals of word count regressed against rank for the 100 most common words in Shakespeare, plotted against the predicted value. Notice that some large predictions have enormous residuals. This plot suggests serious problems with the regression, which you can confirm by looking at Figure 8.11. On the* **right**, *residuals of log word count regressed against log rank for the 100 most common words in Shakespeare, plotted against the predicted value. Notice that residual variance is about the same at each value, a good sign.*

explanatory variables. There is a method for transforming the dependent variable, the **Box-Cox transformation**, that can search for a transformation that improves the regression. This is a one-parameter family of transformations, with parameter $\lambda$. We define the Box-Cox transformation of the dependent variable to be

$$y_i^{(bc)} = \begin{cases} \frac{y_i^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0 \\ \log y_i & \text{if } \lambda = 0 \end{cases} .$$

It turns out to be straightforward to estimate a good value of $\lambda$ using maximum likelihood. One searches for a value of $\lambda$ that makes residuals look most like a normal distribution. Statistical software will do it for you; the exercises sketch out the method. This transformation can produce significant improvements in a regression. Figure **??** shows a small but acceptable improvement in regression of weight against height using the Box-Cox transformation (residuals in Figure 8.13). Figure 8.14 shows an improvement in the regression of weight against all variables for the bodyfat dataset obtained using the Box-Cox transformation. Figure 8.15 shows an improvement in the regression of Boston house price against all variables obtained using the Box-Cox transformation.

**Worked example 8.2** *BoxCox with R*

Regress weight against height using the Box-Cox transformation

**Solution:** This example is mainly used to demonstrate how easy it is to do Box-Cox in R. There is sample code in listing **??**. The line `boxcox(...)` will produce a plot on your screen - you look for the location of the peak, and that's the parameter value to use. In this case, the peak is close to zero, so we should work with the log of the dependent variable. Notice how neat `abline` is — you pass it a linear regression, and it puts a line on a plot.

## 8.3 WHICH VARIABLES ARE IMPORTANT?

### 8.3.1 Regularizing Linear Regressions

One occasionally important difficulty is that the explanatory variables might be significantly correlated. If they are, then it will generally be easy to predict one explanatory variable from another. This means that $\mathcal{X}^T\mathcal{X}$ may have some very small eigenvalues (because there is a vector $\mathbf{u}$ so that $\mathcal{X}\mathbf{u}$ is small; this means that $\mathbf{u}^T\mathcal{X}^T\mathcal{X}\mathbf{u}$ must be small).

These small eigenvalues lead to bad predictions. If $\mathcal{X}^T\mathcal{X}$ has a small eigenvalue, then there is some vector $\mathbf{v}$ such that $\mathcal{X}^T\mathcal{X}\mathbf{v}$ is small, or, equivalently, that the matrix can turn large vectors into small ones; but that means that $(\mathcal{X}^T\mathcal{X})^{-1}$ will turn some small vectors into big ones. In turn, this means that small errors in $\mathbf{y}$ — which are likely inevitable — will result in big errors in $\beta$. This could cause trouble in two ways. If we are looking at $\beta$ to tell which explanatory variables are important, then large errors in $\beta$ will be a problem. And if we are trying to predict

Listing 8.2: R code used for the linear regression example of worked example 2

```
setwd('/users/daf/Current/courses/Probcourse/Regression/RCode/HeightWeight');
#install.packages('gdata')
library(gdata)
#cement slag ash water super coarse fine age strength
bfd<-read.xls('BodyFat.xls')
summary(bfd)
#IDNO BODYFAT DENSITY AGE WEIGHT HEIGHT ADIPOSITY NECK CHEST ABDOMEN
HIP THIGH KNEE ANKLE BICEPS FOREARM WRIST X
sbfd.lm<-lm(WEIGHT~HEIGHT,data=bfd)
plot(bfd[,c("HEIGHT")], bfd[, c("WEIGHT")],
     main="Linear regression of Weight against Height", xlab="Height", ylab="Wei
abline(sbfd.lm)
#36, 39 41 and 42 are the bad ones
cbfd<-bfd[-c(36, 39, 41,42), ]
scbfd<-lm(WEIGHT~HEIGHT,data=cbfd)
plot(cbfd[,c("HEIGHT")], cbfd[, c("WEIGHT")],
     main="Linear regression of Weight against Height, outliers removed", xlab="
abline(scbfd)

library(MASS)
cbfd.lm<-lm(WEIGHT~HEIGHT,data=cbfd)
boxcox(cbfd.lm, plotit=TRUE)
# suggests lambda=0 is best, strongly
cbfd.llm<-lm(log(WEIGHT)~HEIGHT,data=cbfd)
plot(cbfd[,c("HEIGHT")], log(cbfd[, c("WEIGHT")]),
     main="Linear regression of log Weight against Height, outliers removed", xl
abline(cbfd.llm)
```

new $y$ values, we expect that errors in $\beta$ turn into errors in prediction.

An important and useful way to suppress these errors is to regularize, using the same trick we saw in the case of classification. Instead of choosing $\beta$ to minimize

$$\sum_i (y_i - \mathbf{x}_i^T \beta)^2 = (\mathbf{y} - \mathcal{X}\beta)^T (\mathbf{y} - \mathcal{X}\beta)$$

we minimize

$$\sum_i (y_i - \mathbf{x}_i^T \beta)^2 + \lambda \beta^T \beta = (\mathbf{y} - \mathcal{X}\beta)^T (\mathbf{y} - \mathcal{X}\beta) + \lambda \beta^T \beta$$

where $\lambda > 0$ is a constant. We choose $\lambda$ in the same way we used for classification; split the training set into a training piece and a validation piece, train for different values of $\lambda$, and test the resulting regressions on the validation piece. We choose the $\lambda$ that yields the smallest validation error. Notice we could use multiple splits, and average over the splits.

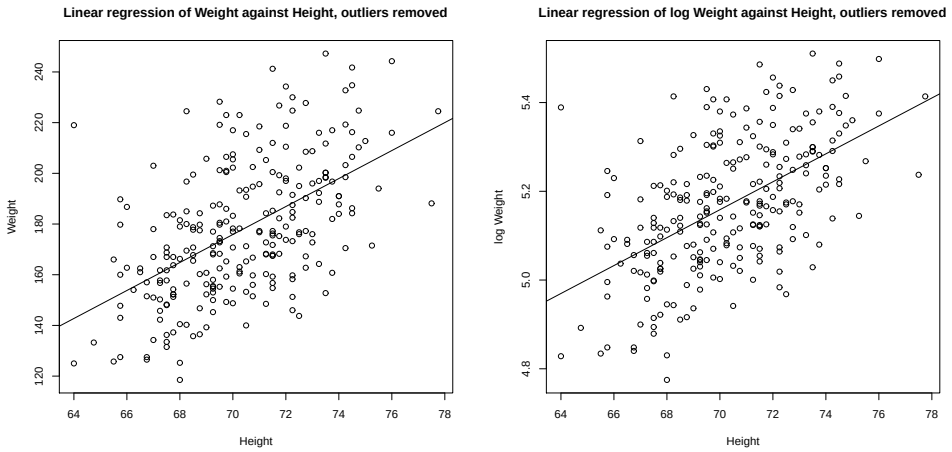


FIGURE 8.12: *On the* **left**, *a linear regression of weight against height for the bodyfat dataset. This is just a copy of the regression of Figure 1, for reference. The* $R^2$ *is 0.29. The Box-Cox method predicts that one should use* $\lambda = 0$ *(equivalently, predict* $\log$ *weight from height). On the* **right**, *a linear regression of log weight against height. The regression is slightly better (*$R^2$ *is 0.3). Figure 1 compares the residuals for these cases.*

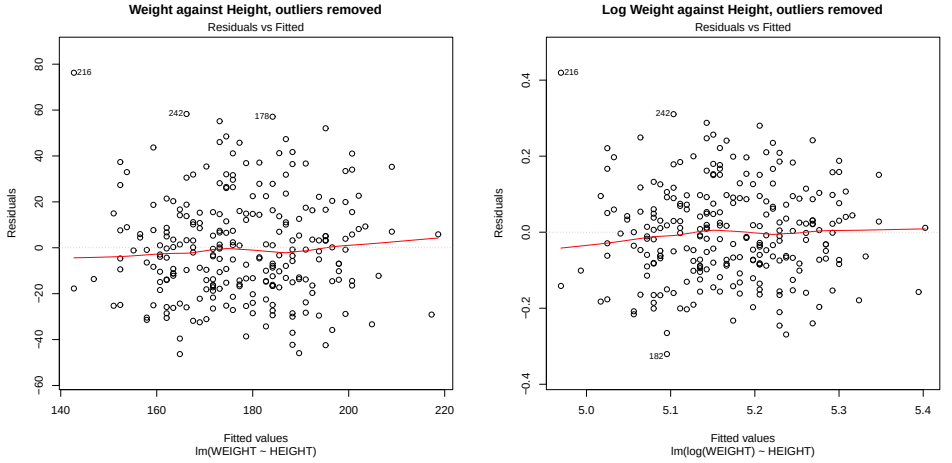


FIGURE 8.13: *On the* **left**, *the residuals against predicted value for the linear regression of weight against height for the bodyfat dataset. This is just a copy of the regression of Figure 1, for reference. The* $R^2$ *is 0.29. The Box-Cox method predicts that one should use* $\lambda = 0$ *(equivalently, predict* $\log$ *weight from height). On the* **right**, *a linear regression of the log of weight against height. The regression is slightly better (*$R^2$ *is 0.3).*

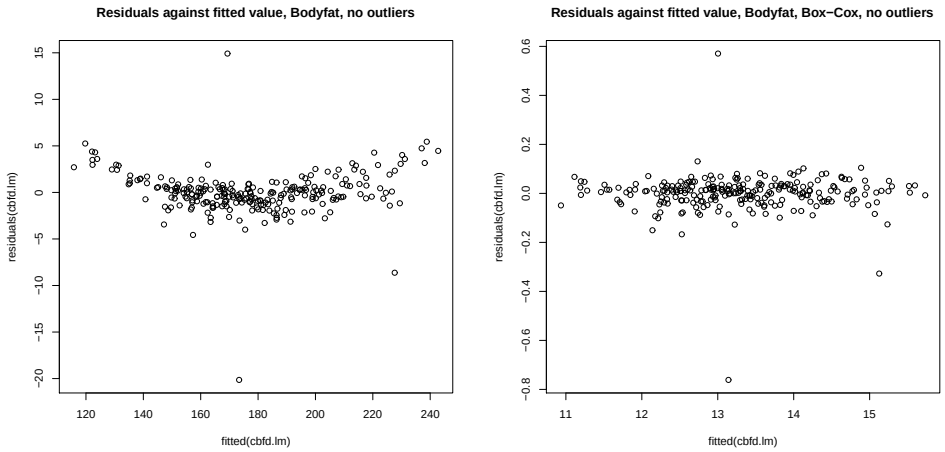


FIGURE 8.14: *On the* **left**, *residuals plotted against predicted value for a regression of weight against all other measurements for the bodyfat dataset, with four outliers removed. Notice that the variance of the residuals changes somewhat as the predicted value increases, giving a banana shape to the data. This suggests that a non-linear transformation of the explanatory variables might be helpful. The Box-Cox procedure suggests* $\lambda = 0.5$*, so we regress* $\sqrt{weight}$ *against the explanatory variables. This yields the plot on the* **right**. *Notice how the banana shape has gone, and the residuals have about the same distribution for each predicted value.*

This helps, because to solve for $\beta$ we must solve the equation

$$(\mathcal{X}^T\mathcal{X} + \lambda\mathcal{I})\beta = \mathcal{X}^T\mathbf{y}$$

(obtained by differentiating with respect to $\beta$ and setting to zero) and the smallest eigenvalue of the matrix $(\mathcal{X}^T\mathcal{X} + \lambda\mathcal{I})$ will be at least $\lambda$.

### 8.3.2 Hypothesis Testing for Regression Coefficients - NEED MATERIAL

### 8.3.3 Which Variables should Contribute? The LASSO


FIGURE 8.15: *On the* **left**, *residuals plotted against predicted value for a regression of Boston house prices against all other measurements (compare Figure 1). The Box-Cox procedure suggests* $\lambda = 0.1$, *so we regress* $weight^{0.1}$ *against the explanatory variables. This yields the plot on the* **right**. *Notice how the residuals have about the same distribution for each predicted value, but the linear structure hasn't gone.*

CHAPTER 9

# Exploiting your Neighbors for Regression

Linear regression predicts $y$ for $\mathbf{x}$ with a simple model, which is quite straightforward to estimate. This model "fills in" or interpolates values between data points. The natural alternative is to collect a really large dataset, and predict using the nearest neighbors. There are cases where this strategy isn't really practical, but if a lot of data is available, it is a very good strategy. It is particularly useful for filling holes in images.

Many different kinds of user want to remove things from images or from video. Art directors might like to remove unattractive telephone wires; restorers might want to remove scratches or marks; there's a long history of government officials removing people with embarrassing politics from publicity pictures (see the fascinating pictures in **?**); and home users might wish to remove a relative they dislike from a family picture. All these users must then find something to put in place of the pixels that were removed. Ideally, a program would create regions of texture that fit in and look convincing, using either other parts of the original image, or other images.

## 9.1 FILLING HOLES IN IMAGES BY MATCHING

### 9.1.1 Filling a Small Hole by Matching Neighborhoods

Imagine I have an image where exactly one pixel value is missing. This pixel could have been erased by noise; it could be a tiny scratch on the film of the image; the disk storage system might have lost it; and so on. I would like to create a value to insert into that pixel's location.

Here is an easy strategy. Take a block of pixels centered on the missing pixel. Find many close matches to this block of pixels in a collection of images, counting only the known pixel values. Each match offers a possible value for the missing pixel. There is no reason to prefer any one value, so choose randomly from the values.

A good measure of similarity between two image neighborhoods can be measured by forming the **sum of squared differences** (or **SSD**) of corresponding pixel values. We assume that the missing pixel is at the center of the patch to be synthesized, which we write $\mathcal{S}$. We assume the patch is square, and adjust the indexes of the patch to run from $-n$ to $n$ in each direction. The sum of squared differences between this patch and an image patch $\mathcal{P}$ of the same size is given by

$$\sum_{(i,j)\in\text{patch},(i,j)\neq(0,0)} (\mathcal{A}_{ij} - \mathcal{B}_{ij})^2.$$

The notation implies that because we don't know the value of the pixel to be

synthesized (which is at $(0,0)$), we don't count it in the sum of squared differences. This similarity value is small when the neighborhoods are similar, and large when they are different (it is essentially the length of the difference vector). However, this measure places the same weight on pixels close to the unknown value as it does on distant pixels. Better results are usually obtained by weighting up nearby pixels and weighting down distant pixels. The function

$$\exp\left(\frac{-(i^2+j^2)}{2\sigma^2}\right)$$

takes the value one when $(i,j)=(0,0)$, and falls off fast as $(i^2+j^2)$ — distance from the origin — grows. This yields **Gaussian weighted SSD**, where the matching cost is

$$\sum_{(i,j)\in\text{patch},(i,j)\neq(0,0)} (\mathcal{A}_{ij}-\mathcal{B}_{ij})^2 \exp\left(\frac{-(i^2+j^2)}{2\sigma^2}\right).$$

Patches with a smaller value of this match cost are better matches.

If we need to fill in only one pixel, then finding the best-matching patch is likely enough. However, we can generalize the idea to fill in more than one pixel. When we do so, it will be important to preserve random structures in the image textures. This means it is a good idea to collect several alternative values for the pixel, and choose one at random. To do so, we find the closest patch. Write $s_{min}$ for the similarity value of the closest patch. We then recover all patches whose similarity value is less than $(1+\epsilon)s_{min}$. Notice that doing so may involve a linear pass through the data (which should be slow, because we should be using a lot of data) or an approximate nearest neighbors query.

We may not need to search a large collection of images to find good patches. Most image textures occupy reasonably large regions in the image, meaning we might find quite good patches matching our patch in the original image itself.

## 9.2 FILLING IN MORE THAN ONE MISSING PIXEL

Now imagine we have many missing pixels. If each is scattered widely across the image, the original procedure will work fine. But things get more interesting if we are missing a block of pixels. In this case, the values of some pixels in the neighborhood of the pixel to be synthesized are not known; these pixels need to be synthesized too. One way to obtain a collection of examples for the pixel of interest is to count only the known values in computing the sum of squared differences, and scale the similarity to take into account the number of known pixels. Write $\mathcal{K}$ for the set of pixels around a point whose values are known, and $\sharp\mathcal{K}$ for the size of this set. We now have, for the similarity function,

$$\frac{1}{\sharp\mathcal{K}} \sum_{(i,j)\in\mathcal{K}} (\mathcal{A}_{ij}-\mathcal{B}_{ij})^2 \exp\left(\frac{-(i^2+j^2)}{2\sigma^2}\right).$$

When there is a hole in the image to be filled, the order in which pixels are filled in is quite important (Figure 1). There are automated methods to manage this. These try to fill in pixels that are near strong boundaries (or are otherwise strongly constrained) first, then fill in other less constrained pixels.

One important application of this procedure is **texture synthesis**, where one takes a small block of texture and expands it to a large block. This is useful, because computer graphics methods and computer games are aggressive consumers of textures, and want to be supplied with very large texture regions. But large examples of sample textures are hard to get — we want to build a large texture from a small one. We could do so by regarding the small texture as part of a larger image, with many missing pixels, yielding Algorithm 9.1.

The size and shape of the neighborhood we use is significant, because it codes the range over which pixels can affect one another's values directly (see Figure 9.3). The original method, due to Efros *et al.* used a square neighborhood, centered at the pixel of interest; current methods do so as well.

```
Choose a texture sample
Insert this sample into the image to be synthesized
Until each location in the image to be synthesized has a value
  For each unsynthesized location on
    the boundary of the block of synthesized values
    Match the neighborhood of this location to the
      example image, ignoring unsynthesized
      locations in computing the matching score
    Choose a value for this location uniformly and at random
      from the set of values of the corresponding locations in the
      matching neighborhoods
  end
end
```

**Algorithm 9.1:** *Non-parametric Texture Synthesis.*

### 9.2.1 Filling Large Holes with Whole Images

If one has a large hole in a large image, we may not be able to just extend a texture to fill the hole. Instead, entire objects might need to appear in the hole (Figure 9.4). There is a straightforward, and extremely effective, way to achieve this. We match the image to a large collection of images. As a match cost, we use the SSD computed for all pixels that we do not want to replace. This yields a set of example images where all the pixels we didn't want to replace are close to those of the query image. From these, we choose one, and fill in the pixels from that image.

There are several ways to choose. If we wish to do so automatically, we could use the example with the smallest distance to the image. Very often, an artist is involved, and then we could prepare a series of alternatives — using, perhaps, the $k$ closest examples — then show them to the artist, who will choose one. This method, which is very simple to describe, is extremely effective in practice.

Filling in the pixels requires some care. One does not usually get the best results by just copying the missing pixels from the matched image into the hole. Instead, it is better to look for a good **seam**. We search for a curve enclosing the

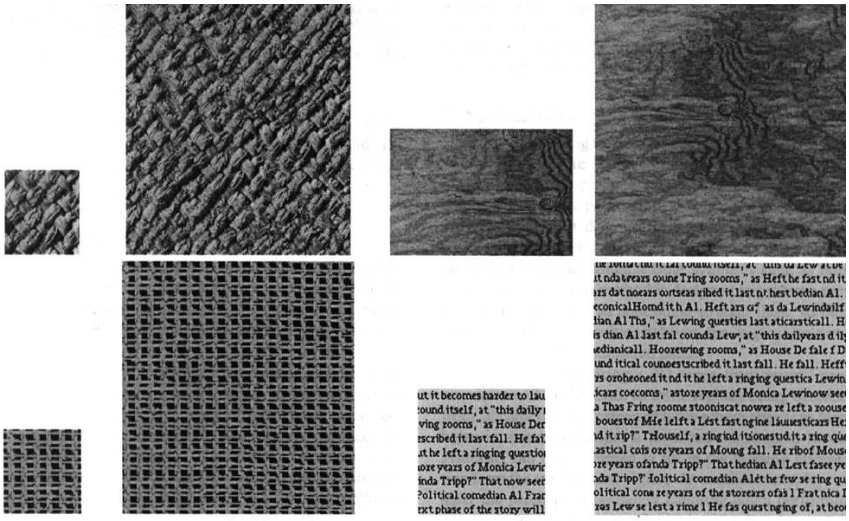

FIGURE 9.1: *? synthesize textures by matching neighborhoods of the image being synthesized to the example image, and then choosing at random amongst the possible values reported by matching neighborhoods (Algorithm 9.1). This means that the algorithm can reproduce complex spatial structures, as these examples indicate. The small block on the* **left** *is the example texture; the algorithm synthesizes the block on the* **right**. *Note that the synthesized text looks like text: it appears to be constructed of words of varying lengths that are spaced like text, and each word looks as though it is composed of letters (though this illusion fails as one looks closely).* This figure was originally published as Figure 3 of "Texture Synthesis by Non-parametric Sampling," A. Efros and T.K. Leung, Proc. IEEE ICCV, 1999 © IEEE, 1999.

missing pixels which (a) is reasonably close to the boundary of the missing pixels and (b) gives a good boundary between the two images. A good boundary is one where the query image (on one side) is "similar to" the matched image (on the other side). A good sense of similarity requires that pixels match well, and that image gradients crossing the boundary tend to match too.

## 9.3 REGRESSION USING NEIGHBORS

In the linear regression model, I did not explain how to regress objects with complicated structure against explanatory variables. This is a problem that arises often in practice. For example, I might want to predict a parse tree (the object with complicated structure) from a sentence (the explanatory variables). As another example, I might want to predict a map of the shadows in an image (the object with complicated structure) against an image (the explanatory variables). As yet another example, I might want to predict which direction to move the controls on a radio-controlled helicopter (the object with complicated structure) against a path plan and the current state of the helicopter (the explanatory variables). In each case, it seems unlikely that a pure linear regression is likely to resolve the problem.

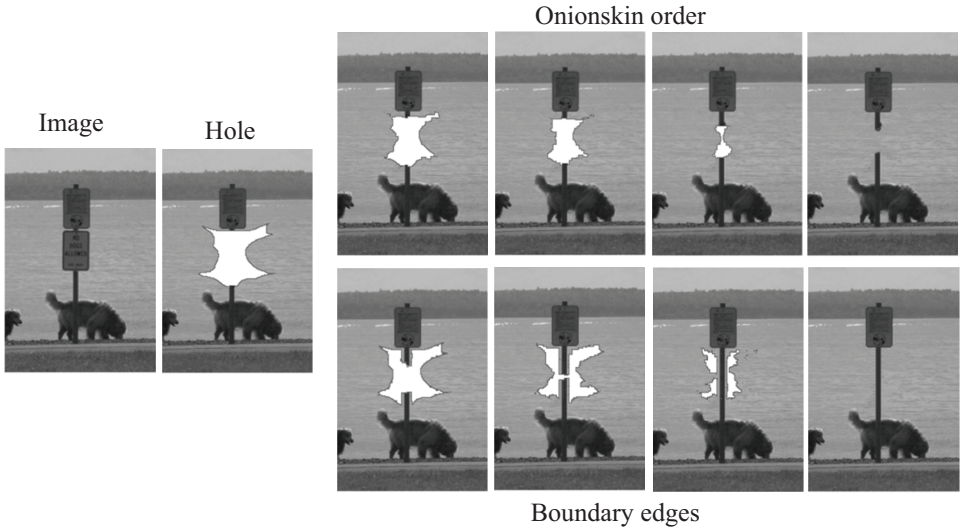


FIGURE 9.2: *Texture synthesis methods can fill in holes accurately, but the order in which pixels are synthesized is important. In this figure, we wish to remove the sign, while preserving the signpost. Generally, we want to fill in pixels where most of the neighbors are known first. This yields better matching patches. One way to do so is to fill in from the boundary. However, if we simply work our way inwards (onionskin filling), long scale image structures tend to disappear. It is better to fill in patches close to edges first.* This figure was originally published as Figure 11 of "Region Filling and Object Removal by Exemplar-Based Image Inpainting," by A. Criminisi, P. Perez, and K. Toyama, IEEE Transactions on Image Processing, 2004 © IEEE, 2004.

Looking at neighbors is a very good way to solve such problems. The general strategy is relatively simple. We find a large collection of pairs of training data. Write $\mathbf{x}_i$ for the explanatory variables for the $i$'th example, and $\mathbf{y}_i$ for the dependent variable in the $i$'th example. This dependent variable could be anything — it doesn't need to be a single number. It might be a tree, or a shadow map, or a word, or anything at all. I wrote it as a vector because I needed to choose some notation.

In the simplest, and most general, approach, we obtain a prediction for a new set of explanatory variables $\mathbf{x}$ by (a) finding the nearest neighbor and then (b) producing the dependent variable for that neighbor. We might vary the strategy slightly by using an approximate nearest neighbor. You have already seen this strategy in action filling holes in images.

If the dependent variables have enough structure that it is possible to summarize a collection of different dependent variables, then we might recover the $k$ nearest neighbors and summarize their dependent variables. How we summarize rather depends on the dependent variables. For example, it is a bit difficult to imagine the average of a set of trees, but quite straightforward to average numbers.

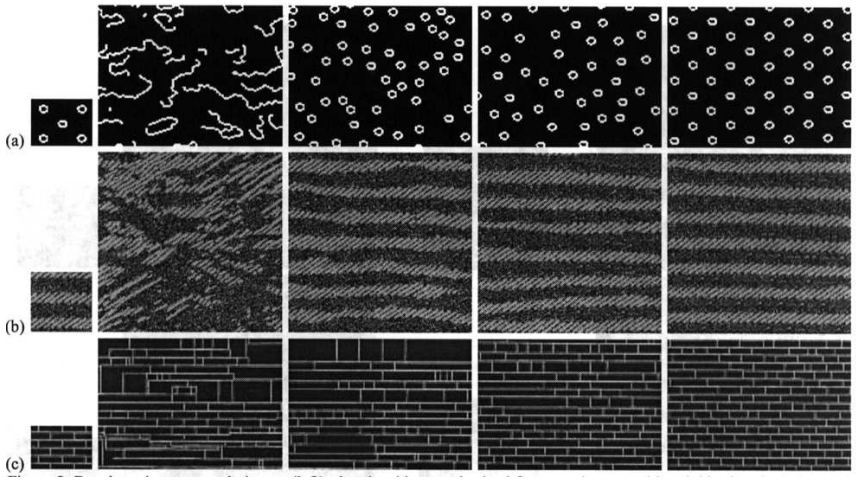


FIGURE 9.3: *The size of the image neighborhood to be matched makes a significant difference in Algorithm 9.1. In the figure, the textures at the right are synthesized from the small blocks on the* **left**, *using neighborhoods that are increasingly large as one moves to the* **right**. *If very small neighborhoods are matched, then the algorithm cannot capture large-scale effects easily. For example, in the case of the spotty texture, if the neighborhood is too small to capture the spot structure (and so sees only pieces of curve), the algorithm synthesizes a texture consisting of curve segments. As the neighborhood gets larger, the algorithm can capture the spot structure, but not the even spacing. With very large neighborhoods, the spacing is captured as well.* This figure was originally published as Figure 2 of "Texture Synthesis by Non-parametric Sampling," A. Efros and T.K. Leung, Proc. IEEE ICCV, 1999 © IEEE, 1999.

If the dependent variable was a word, we might not be able to average words, but we can vote and choose the most popular word. If the dependent variable is a vector, we can compute either distance weighted averages or a distance weighted linear regression.

### 9.3.1 Distance Weighted Averages

### 9.3.2 Distance Weighted Linear Regressions

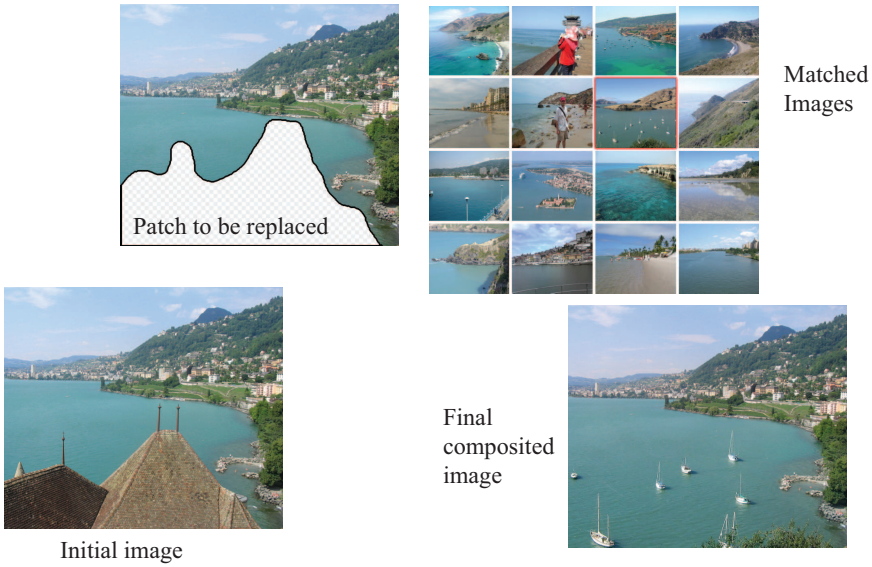


FIGURE 9.4: *We can fill large holes in images by matching the image to a collection, choosing one element of the collection, then cutting out an appropriate block of pixels and putting them into the hole in the query image. In this case, the hole has been made by an artist, who wishes to remove the roofline from the view. Notice how there are a range of objects (boats, water) that have been inserted into the hole. These objects are a reasonable choice, because the overall structures of the query and matched image are largely similar — getting an image that matches most of the query image supplies enough context to ensure that the rest "makes sense".*

CHAPTER 10

# Models of High Dimensional Data

High-dimensional data comes with problems. Data points tend not to be where you think; they can scattered quite far apart, and can be quite far from the mean. Except in special cases, the only really reliable probability model is the Gaussian (or Gaussian blob, or blob).

There is an important rule of thumb for coping with high dimensional data: **Use simple models**. A blob is another good, simple model. Modelling data as a blob involves computing its mean and its covariance. Sometimes, as we shall see, the covariance can be hard to compute. Even so, a blob model is really useful. It is natural to try and extend this model to cover datasets that don't obviously consist of a single blob.

One very good, very simple, model for high dimensional data is to assume that it consists of multiple blobs. To build models like this, we must determine (a) what the blob parameters are and (b) which datapoints belong to which blob. Generally, we will collect together data points that are close and form blobs out of them. This process is known as **clustering**.

Clustering is a somewhat puzzling activity. It is extremely useful to cluster data, and it seems to be quite important to do it reasonably well. But it surprisingly hard to give crisp criteria for a good (resp. bad) clustering of a dataset. Usually, clustering is part of building a model, and the main way to know that the clustering algorithm is bad is that the model is bad.

## 10.1 THE CURSE OF DIMENSION

High dimensional models display uninituitive behavior (or, rather, it can take years to make your intuition see the true behavior of high-dimensional models as natural). In these models, most data lies in places you don't expect. We will do several simple calculations with an easy high-dimensional distribution to build some intuition.

Assume our data lies within a cube, with edge length two, centered on the origin. This means that each component of $\mathbf{x}_i$ lies in the range $[-1, 1]$. One simple model for such data is to assume that each dimension has uniform probability density in this range. In turn, this means that $P(x) = \frac{1}{2^d}$. The mean of this model is at the origin, which we write as $\mathbf{0}$.

The first surprising fact about high dimensional data is that most of the data can lie quite far away from the mean. For example, we can divide our dataset into two pieces. $\mathcal{A}(\epsilon)$ consists of all data items where *every* component of the data has a value in the range $[-(1-\epsilon), (1-\epsilon)]$. $\mathcal{B}(\epsilon)$ consists of all the rest of the data. If you think of the data set as forming a cubical orange, then $\mathcal{B}(\epsilon)$ is the rind (which has thickness $\epsilon$) and $\mathcal{A}(\epsilon)$ is the fruit.

Your intuition will tell you that there is more fruit than rind. This is true, for three dimensional oranges, but not true in high dimensions. The fact that the orange is cubical just simplifies the calculations, but has nothing to do with the

real problem.

We can compute $P(\{\mathbf{x} \in \mathcal{A}(\epsilon)\})$ and $P(\{\mathbf{x} \in \mathcal{A}(\epsilon)\})$. These probabilities tell us the probability a data item lies in the fruit (resp. rind). $P(\{\mathbf{x} \in \mathcal{A}(\epsilon)\})$ is easy to compute as

$$P(\{\mathbf{x} \in \mathcal{A}(\epsilon)\}) = (2(1-\epsilon)))^d \left(\frac{1}{2^d}\right) = (1-\epsilon)^d$$

and

$$P(\{\mathbf{x} \in \mathcal{B}(\epsilon)\}) = 1 - P(\{\mathbf{x} \in \mathcal{A}(\epsilon)\}) = 1 - (1-\epsilon)^d.$$

But notice that, as $d \to \infty$,

$$P(\{\mathbf{x} \in \mathcal{A}(\epsilon)\}) \to 0.$$

This means that, for large $d$, we expect most of the data to be in $\mathcal{B}(\epsilon)$. Equivalently, for large $d$, we expect that at least one component of each data item is close to either 1 or $-1$.

This suggests (correctly) that much data is quite far from the origin. It is easy to compute the average of the squared distance of data from the origin. We want

$$\mathbb{E}\left[\mathbf{x}^T\mathbf{x}\right] = \int_{\text{box}} \left(\sum_i x_i^2\right) P(\mathbf{x}) d\mathbf{x}$$

but we can rearrange, so that

$$\mathbb{E}\left[\mathbf{x}^T\mathbf{x}\right] = \sum_i \mathbb{E}\left[x_i^2\right] = \sum_i \int_{\text{box}} x_i^2 P(\mathbf{x}) d\mathbf{x}.$$

Now each component of $\mathbf{x}$ is independent, so that $P(\mathbf{x}) = P(x_1)P(x_2)\dots P(x_d)$. Now we substitute, to get

$$\mathbb{E}\left[\mathbf{x}^T\mathbf{x}\right] = \sum_i \mathbb{E}\left[x_i^2\right] = \sum_i \int_{-1}^{1} x_i^2 P(x_i) dx_i = \sum_i \frac{1}{2} \int_{-1}^{1} x_i^2 dx_i = \frac{d}{3},$$

so as $d$ gets bigger, most data points will be further and further from the origin. Worse, as $d$ gets bigger, data points tend to get further and further from one another. We can see this by computing the average of the squared distance of data points from one another. Write $\mathbf{u}$ for one data point and $\mathbf{v}$; we can compute

$$\mathbb{E}\left[d(\mathbf{u},\mathbf{v})^2\right] = \int_{\text{box}} \int_{\text{box}} \sum_i (u_i - v_i)^2 d\mathbf{u} d\mathbf{v} = \mathbb{E}\left[\mathbf{u}^T\mathbf{u}\right] + \mathbb{E}\left[\mathbf{v}^T\mathbf{v}\right] - \mathbb{E}\left[\mathbf{u}^T\mathbf{v}\right]$$

but since $\mathbf{u}$ and $\mathbf{v}$ are independent, we have $\mathbb{E}\left[\mathbf{u}^T\mathbf{v}\right] = \mathbb{E}[\mathbf{u}]^T\mathbb{E}[\mathbf{v}] = 0$. This yields

$$\mathbb{E}\left[d(\mathbf{u},\mathbf{v})^2\right] = 2\frac{d}{3}$$

meaning that, for large $d$, we expect our data points to be quite far apart.

It is difficult to build histogram representations for high dimensional data. The strategy of dividing the domain into boxes, then counting data into them, fails miserably because there are too many boxes. In the case of our cube, imagine we wish to divide each dimension in half (i.e. between $[-1, 0]$ and between $[0, 1]$). Then we must have $2^d$ boxes. This presents two problems. First, we will have difficulty representing this number of boxes. Second, unless we are exceptionally lucky, most boxes must be empty because we will not have $2^d$ data items.

However, one representation is extremely effective. We can represent data as a collection of **clusters** — coherent blobs of similar datapoints that could, under appropriate circumstances, be regarded as the same. We discuss how to build these clusters, then what to do with them.

## 10.2 AGGLOMERATIVE AND DIVISIVE CLUSTERING

There are two natural algorithms for clustering. In **divisive clustering**, the entire data set is regarded as a cluster, and then clusters are recursively split to yield a good clustering (Algorithm 10.2). In **agglomerative clustering**, each data item is regarded as a cluster, and clusters are recursively merged to yield a good clustering (Algorithm 10.1).

```
Make each point a separate cluster
Until the clustering is satisfactory
    Merge the two clusters with the
      smallest inter-cluster distance
end
```

**Algorithm 10.1:** *Agglomerative Clustering or Clustering by Merging.*

```
Construct a single cluster containing all points
Until the clustering is satisfactory
    Split the cluster that yields the two
      components with the largest inter-cluster distance
end
```

**Algorithm 10.2:** *Divisive Clustering, or Clustering by Splitting.*

There are two major issues in thinking about clustering:

- *What is a good inter-cluster distance?* Agglomerative clustering uses an inter-cluster distance to fuse nearby clusters; divisive clustering uses it to split insufficiently coherent clusters. Even if a natural distance between data points is available (which might not be the case for vision problems), there is no canonical inter-cluster distance. Generally, one chooses a distance that seems appropriate for the data set. For example, one might choose the distance between the closest elements as the inter-cluster distance, which tends to

yield extended clusters (statisticians call this method **single-link clustering**). Another natural choice is the maximum distance between an element of the first cluster and one of the second, which tends to yield rounded clusters (statisticians call this method **complete-link clustering**). Finally, one could use an average of distances between elements in the cluster, which also tends to yield "rounded" clusters (statisticians call this method **group average clustering**).

- *How many clusters are there?* This is an intrinsically difficult task if there is no model for the process that generated the clusters. The algorithms we have described generate a hierarchy of clusters. Usually, this hierarchy is displayed to a user in the form of a **dendrogram**—a representation of the structure of the hierarchy of clusters that displays inter-cluster distances—and an appropriate choice of clusters is made from the dendrogram (see the example in Figure 10.1).

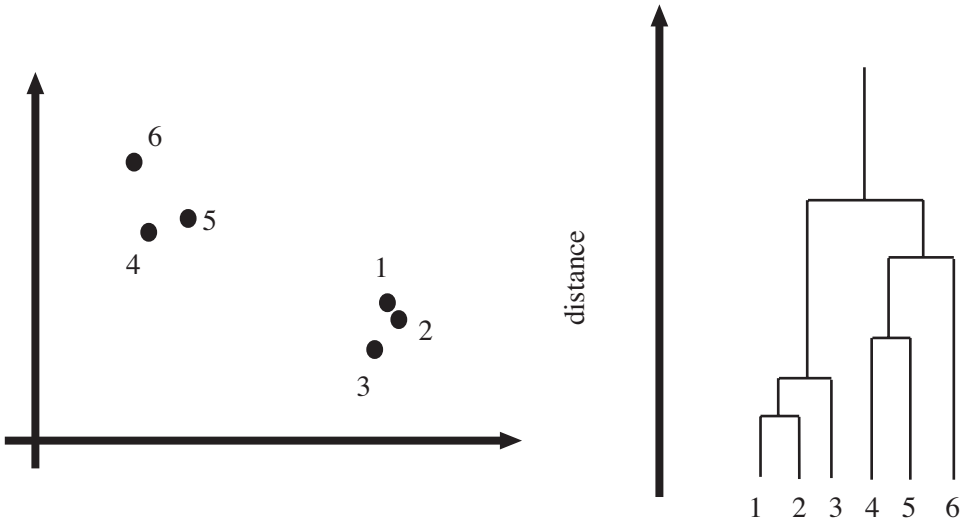


FIGURE 10.1: **Left**, *a data set;* **right**, *a dendrogram obtained by agglomerative clustering using single-link clustering. If one selects a particular value of distance, then a horizontal line at that distance splits the dendrogram into clusters. This representation makes it possible to guess how many clusters there are and to get some insight into how good the clusters are.*

The main difficulty in using a divisive model is knowing where to split. This is sometimes made easier for particular kinds of data. For example, we could segment an image by clustering pixel values. In this case, you can sometimes find good splits by constructing a histogram of intensities, or of color values.

Another important thing to notice about clustering from the example of figure 10.1 is that there is no right answer. There are a variety of different clusterings of the same data. For example, depending on what scales in that figure mean, it

might be right to zoom out and regard all of the data as a single cluster, or to zoom in and regard each data point as a cluster. Each of these representations may be useful.

### 10.2.1 Clustering and Distance

In the algorithms above, and in what follows, we assume that the features are scaled so that distances (measured in the usual way) between data points are a good representation of their similarity. This is quite an important point. For example, imagine we are clustering data representing brick walls. The features might contain several distances: the spacing between the bricks, the length of the wall, the height of the wall, and so on. If these distances are given in the same set of units, we could have real trouble. For example, assume that the units are centimeters. Then the spacing between bricks is of the order of one or two centimeters, but the heights of the walls will be in the hundreds of centimeters. In turn, this means that the distance between two datapoints is likely to be completely dominated by the height and length data. This could be what we want, but it might also not be a good thing.

There are some ways to manage this issue. One is to know what the features measure, and know how they should be scaled. Usually, this happens because you have a deep understanding of your data. If you don't (which happens!), then it is often a good idea to try and normalize the scale of the data set. There are two good strategies. The simplest is to translate the data so that it has zero mean (this is just for neatness - translation doesn't change distances), then scale each direction so that it has unit variance. More sophisticated is to translate the data so that it has zero mean, then transform it so that each direction is independent and has unit variance. Doing so is sometimes referred to as **decorrelation** or **whitening** (because you make the data more like white noise).

**TODO:** no - fix the whitening link here

**TODO:** show how in R?

**TODO:** worked example in R?

### 10.2.2 Example: Agglomerative Clustering

Matlab provides some tools that are useful for agglomerative clustering. These functions use a scheme where one first builds the whole tree of merges, then analyzes that tree to decide which clustering to report. `linkage` will determine which pairs of clusters should be merged at which step (there are arguments that allow you to choose what type of inter-cluster distance it should use); `dendrogram` will plot you a dendrogram; and `cluster` will extract the clusters from the linkage, using a variety of options for choosing the clusters. I used these functions to prepare the dendrogram of figure 10.2 for the height-weight dataset of section **??** (from `http://www2.stetson.edu/~jrasp/data.htm`; look for bodyfat.xls). I deliberately forced Matlab to plot the whole dendrogram, which accounts for the crowded look of the figure (you can allow it to merge small leaves, etc.). I used a single-link strategy.

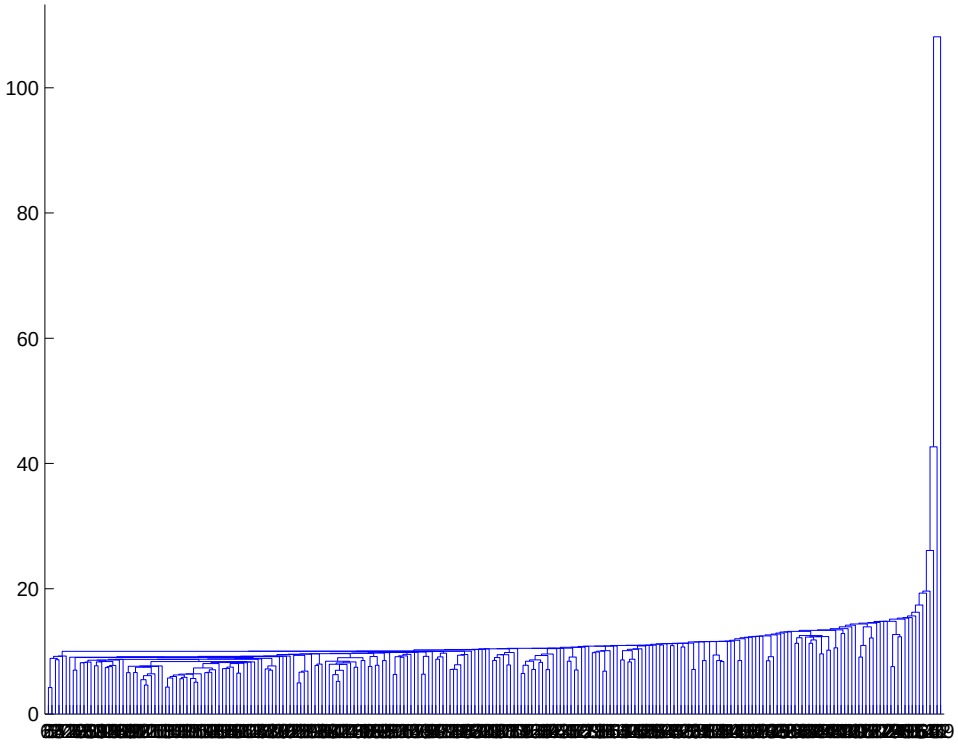


FIGURE 10.2: *A dendrogram obtained from the body fat dataset, using single link clustering. Recall that the data points are on the horizontal axis, and that the vertical axis is distance; there is a horizontal line linking two clusters that get merged, established at the height at which they're merged. I have plotted the entire dendrogram, despite the fact it's a bit crowded at the bottom, because it shows that most data points are relatively close (i.e. there are lots of horizontal branches at about the same height).*

In particular, notice that many data points are about the same distance from one another, which suggests a single big cluster with a smaller set of nearby clusters. The clustering of figure 10.3 supports this view. I plotted the data points on the first two principal components, using different colors and shapes of marker to indicate different clusters. There are a total of 30 clusters here, though most are small.

As another example, I obtained the seed dataset from the UC Irvine Machine Learning Dataset Repository (you can find it at `http://archive.ics.uci.edu/ml/datasets/seeds`). Each item consists of seven measurements of a wheat kernel; there are three types of wheat represented in this dataset. As you can see in figures 10.4 and 10.5, this data clusters rather well.

## 10.3 THE K-MEANS ALGORITHM AND VARIANTS

**TODO:** Figure notes: Iris figure, estimate of k, 2d iris plot, projected onto pca

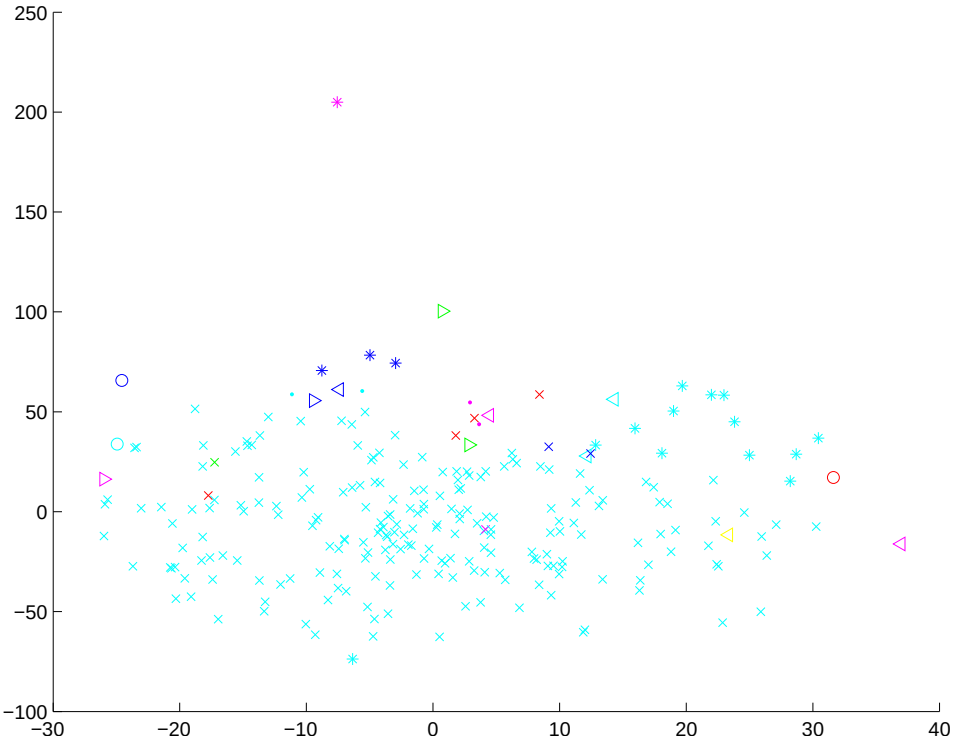


FIGURE 10.3: *A clustering of the body fat dataset, using agglomerative clustering, single link distance, and requiring a maximum of 30 clusters. I have plotted each cluster with a distinct marker (though some markers differ only by color; you might need to look at the PDF version to see this figure at its best). Notice that one cluster contains much of the data, and that there are a set of small isolated clusters. The original data is 16 dimensional, which presents plotting problems; I show a scatter plot on the first two principal components (though I computed distances for clustering in the original 16 dimensional space).*

plot, another cluster dataset

Assume we have a dataset that, we believe, forms many clusters that look like blobs. If we knew where the center of each of the clusters was, it would be easy to tell which cluster each data item belonged to — it would belong to the cluster with the closest center. Similarly, if we knew which cluster each data item belonged to, it would be easy to tell where the cluster centers were — they'd be the mean of the data items in the cluster. This is the point closest to every point in the cluster.

We can turn these observations into an algorithm. Assume that we know how many clusters there are in the data, and write $k$ for this number. The $j$th data item to be clustered is described by a feature vector $\mathbf{x}_j$. We write $\mathbf{c}_i$ for the center of the $i$th cluster. We assume that most data items are close to the center of their

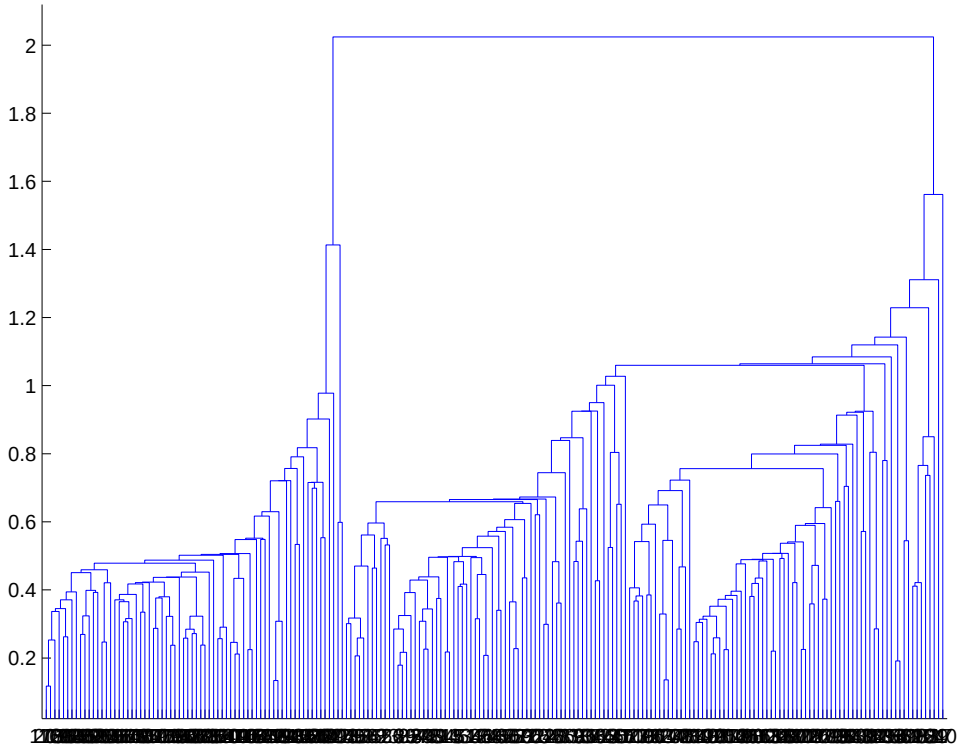


FIGURE 10.4: *A dendrogram obtained from the seed dataset, using single link clustering. Recall that the data points are on the horizontal axis, and that the vertical axis is distance; there is a horizontal line linking two clusters that get merged, established at the height at which they're merged. I have plotted the entire dendrogram, despite the fact it's a bit crowded at the bottom, because you can now see how clearly the data set clusters into a small set of clusters — there are a small number of vertical "runs".*

cluster. This suggests that we cluster the data by minimizing the the cost function

$$\Phi(\text{clusters}, \text{data}) = \sum_{i \in \text{clusters}} \left\{ \sum_{j \in i\text{th cluster}} (\mathbf{x}_j - \mathbf{c}_i)^T (\mathbf{x}_j - \mathbf{c}_i) \right\}.$$

Notice that if we know the center for each cluster, it is easy to determine which cluster is the best choice for each point. Similarly, if the allocation of points to clusters is known, it is easy to compute the best center for each cluster. However, there are far too many possible allocations of points to clusters to search this space for a minimum. Instead, we define an algorithm that iterates through two activities:

- Assume the cluster centers are known and, allocate each point to the closest cluster center.

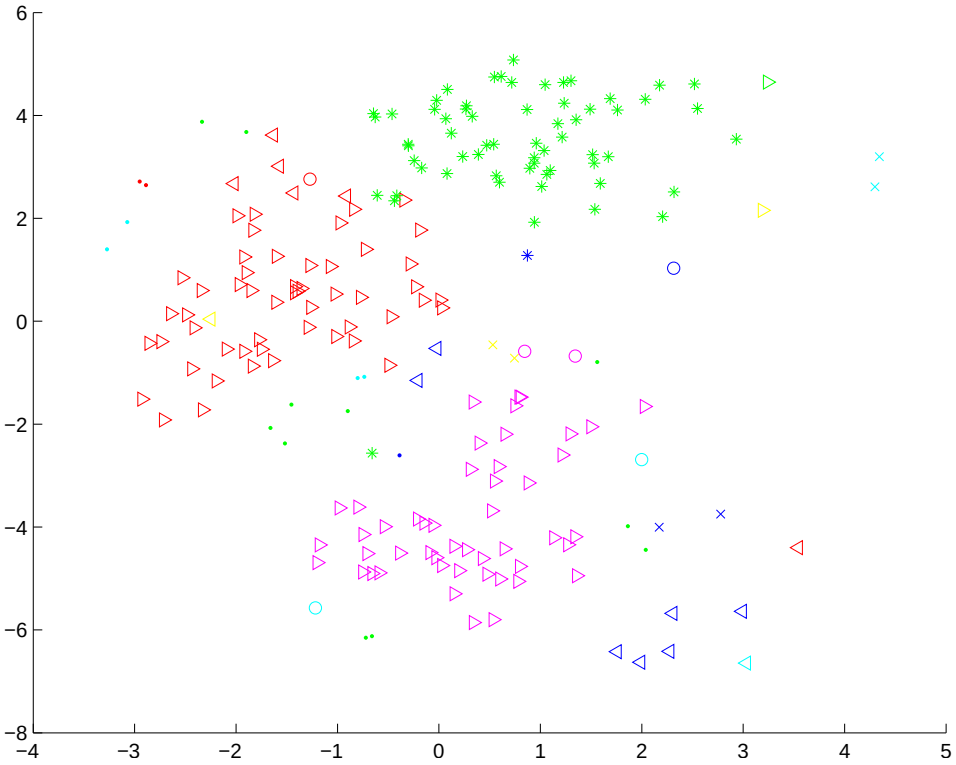


FIGURE 10.5: *A clustering of the seed dataset, using agglomerative clustering, single link distance, and requiring a maximum of 30 clusters. I have plotted each cluster with a distinct marker (though some markers differ only by color; you might need to look at the PDF version to see this figure at its best). Notice that there are a set of fairly natural isolated clusters. The original data is 8 dimensional, which presents plotting problems; I show a scatter plot on the first two principal components (though I computed distances for clustering in the original 8 dimensional space).*

- Assume the allocation is known, and choose a new set of cluster centers. Each center is the mean of the points allocated to that cluster.

We then choose a start point by randomly choosing cluster centers, and then iterate these stages alternately. This process eventually converges to a local minimum of the objective function (the value either goes down or is fixed at each step, and it is bounded below). It is not guaranteed to converge to the global minimum of the objective function, however. It is also not guaranteed to produce $k$ clusters, unless we modify the allocation phase to ensure that each cluster has some nonzero number of points. This algorithm is usually referred to as **k-means** (summarized in Algorithm 10.3).

Usually, we are clustering high dimensional data, so that visualizing clusters can present a challenge. If the dimension isn't too high, then we can use panel plots. An alternative is to project the data onto two principal components, and plot the

```
Choose k data points to act as cluster centers
Until the cluster centers change very little
    Allocate each data point to cluster whose center is nearest.
    Now ensure that every cluster has at least
      one data point; one way to do this is by
      supplying empty clusters with a point chosen at random from
      points far from their cluster center.
    Replace the cluster centers with the mean of the elements
      in their clusters.
end
```

**Algorithm 10.3:** *Clustering by K-Means.*

clusters there. A natural dataset to use to explore k-means is the iris data, where we know that the data should form three clusters (because there are three species). Recall this dataset from section **??**. I reproduce figure 4.5 from that section, for comparison. Figures 10.6, **??** and **??** show different k-means clusterings of that data.

### 10.3.1 How to choose K

The iris data is just a simple example. We know that the data forms clean clusters, and we know there should be three of them. Usually, we don't know how many clusters there should be, and we need to choose this by experiment. One strategy is to cluster for a variety of different values of $k$, then look at the value of the cost function for each. If there are more centers, each data point can find a center that is close to it, so we expect the value to go down as $k$ goes up. This means that looking for the $k$ that gives the smallest value of the cost function is not helpful, because that $k$ is always the same as the number of data points (and the value is then zero). However, it can be very helpful to plot the value as a function of $k$, then look at the "knee" of the curve. Figure 10.9 shows this plot for the iris data. Notice that $k = 3$ — the "true" answer — doesn't look particularly special, but $k = 2$, $k = 3$, or $k = 4$ all seem like reasonable choices. It is possible to come up with a procedure that makes a more precise recommendation by penalizing clusterings that use a large $k$, because they may represent inefficient encodings of the data. However, this is often not worth the bother.

In some special cases (like the iris example), we might know the right answer to check our clustering against. In such cases, one can evaluate the clustering by looking at the number of different labels in a cluster (sometimes called the purity), and the number of clusters. A good solution will have few clusters, all of which have high purity.

Mostly, we don't have a right answer to check against. An alternative strategy, which might seem crude to you, for choosing $k$ is extremely important in practice. Usually, one clusters data to use the clusters in an application (one of the most important, vector quantization, is described in section 10.4). There are usually

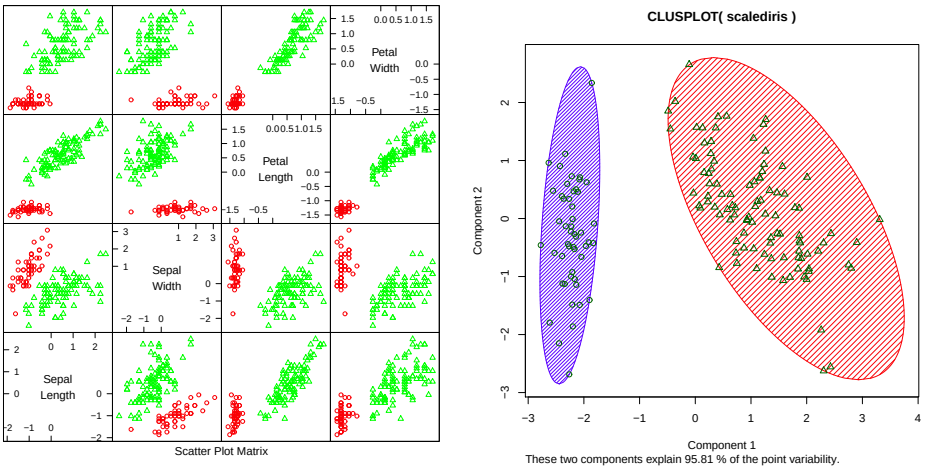


FIGURE 10.6: *On the* **left**, *a panel plot of the iris data clustered using k-means with* $k = 2$. *By comparison with figure 10.9, notice how the* versicolor *and* verginica *clusters appear to have been merged. On the* **right**, *this data set projected onto the first two principal components, with one blob drawn over each cluster.*

natural ways to evaluate this application. For example, vector quantization is often used as an early step in texture recognition or in image matching; here one can evaluate the error rate of the recognizer, or the accuracy of the image matcher. One then chooses the $k$ that gets the best evaluation score on validation data. In this view, the issue is not how good the clustering is; it's how well the system that uses the clustering works.

### 10.3.2 Soft Assignment

One difficulty with k-means is that each point must belong to exactly one cluster. But, given we don't know how many clusters there are, this seems wrong. If a point is close to more than one cluster, why should it be forced to choose? This reasoning suggests we assign points to cluster centers with weights.

We allow each point to carry a total weight of 1. In the conventional k-means algorithm, it must choose a single cluster, and assign its weight to that cluster alone. In soft-assignment k-means, the point can allocate some weight to each cluster center, as long as (a) the weights are all non-negative and (b) the weights sum to one. Write $w_{i,j}$ for the weight connecting point $i$ to cluster center $j$. We interpret these weights as the degree to which the point participates in a particular cluster. We require $w_{i,j} \geq 0$ and $\sum_j w_{i,j} = 1$.

We would like $w_{i,j}$ to be large when $\mathbf{x}_i$ is close to $\mathbf{c}_j$, and small otherwise. Write $d_{i,j}$ for the distance $\| \mathbf{x}_i - \mathbf{c}_j \|$ between these two. Write

$$s_{i,j} = e^{\frac{-d_{i,j}^2}{2\sigma^2}}$$

where $\sigma > 0$ is a choice of scaling parameter. This is often called the affinity

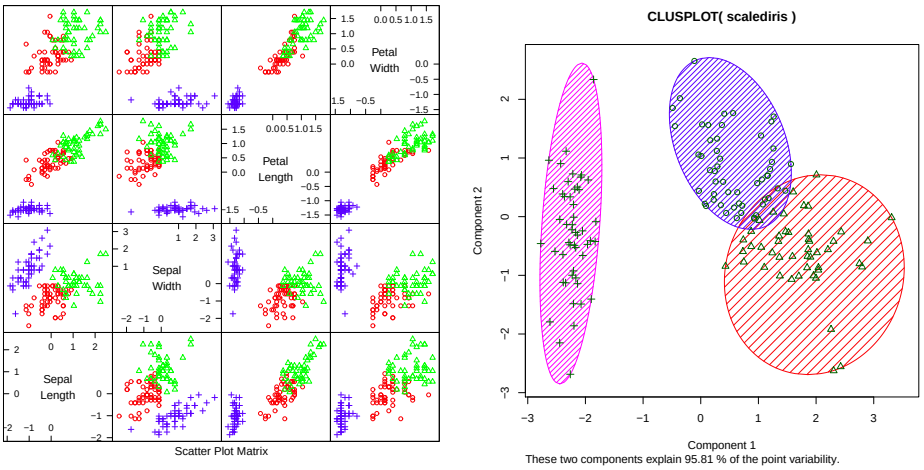


FIGURE 10.7: *On the* **left**, *a panel plot of the iris data clustered using k-means with* $k = 3$. *By comparison with figure 10.9, notice how the clusters appear to follow the species labels. On the* **right**, *this data set projected onto the first two principal components, with one blob drawn over each cluster.*

between the point $i$ and the center $j$. Now a natural choice of weights is

$$w_{i,j} = \frac{s_{i,j}}{\sum_{l=1}^{k} s_{i,l}}.$$

All these weights are non-negative, they sum to one, and the weight is large if the point is much closer to one center than to any other. The scaling parameter $\sigma$ sets the meaning of "much closer" — we measure distance in units of $\sigma$.

Once we have weights, re-estimating the cluster centers is easy. We use a weights to compute a weighted average of the points. In particular, we re-estimate the $j$'th cluster center by

$$\frac{\sum_i w_{i,j}\mathbf{x}_i}{\sum_i w_{i,j}}.$$

Notice that k-means is a special case of this algorithm where $\sigma$ limits to zero. In this case, each point has a weight of one for some cluster, and zero for all others, and the weighted mean becomes an ordinary mean. I have collected the description into Algorithm 10.4 for convenience.

### 10.3.3 General Comments on K-Means

If you experiment with k-means, you will notice one irritating habit of the algorithm. It almost always produces either some rather spread out clusters, or some single element clusters. Most clusters are usually rather tight and blobby clusters, but there is usually one or more bad cluster. This is fairly easily explained. Because every data point must belong to some cluster, data points that are far from all others (a) belong to some cluster and (b) very likely "drag" the cluster center into

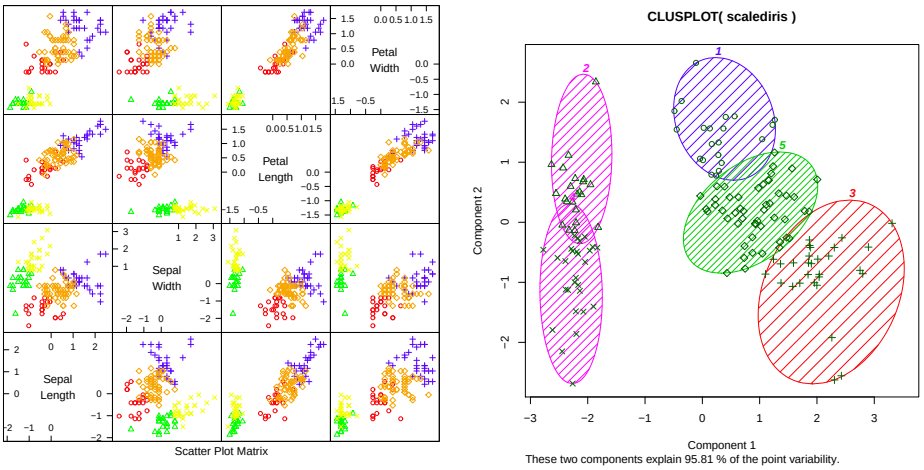


FIGURE 10.8: *On the* **left**, *a panel plot of the iris data clustered using k-means with* $k = 5$. *By comparison with figure 10.9, notice how* setosa *seems to have been broken in two groups, and* versicolor *and* verginica *into a total of three . On the* **right**, *this data set projected onto the first two principal components, with one blob drawn over each cluster.*

a poor location. This applies even if you use soft assignment, because every point must have total weight one. If the point is far from all others, then it will be assigned to the closest with a weight very close to one, and so may drag it into a poor location, or it will be in a cluster on its own.

There are ways to deal with this. If $k$ is very big, the problem is often not significant, because then you simply have many single element clusters that you can ignore. It isn't always a good idea to have too large a $k$, because then some larger clusters might break up. An alternative is to have a junk cluster. Any point that is too far from the closest true cluster center is assigned to the junk cluster, and the center of the junk cluster is not estimated. Notice that points should not be assigned to the junk cluster permanently; they should be able to move in and out of the junk cluster as the cluster centers move.

In some cases, we want to cluster objects that can't be averaged. For example, you can compute distances between two trees but you can't meaningfully average them. In some cases, you might have a table of distances between objects, but not know vectors representing the objects. For example, one could collect data on the similarities between countries (as in Section **??**, particularly Figure **??**), then try and cluster using this data (similarities can be turned into distances by, for example, taking the negative logarithm). A variant of k-means, known as k-medoids, applies to this case.

In k-medoids, the cluster centers are data items rather than averages, but the rest of the algorithm has a familiar form. We assume the number of medoids is known, and initialize these randomly. We then iterate two procedures. In the first, we allocate data points to medoids. In the second, we choose the best medoid for

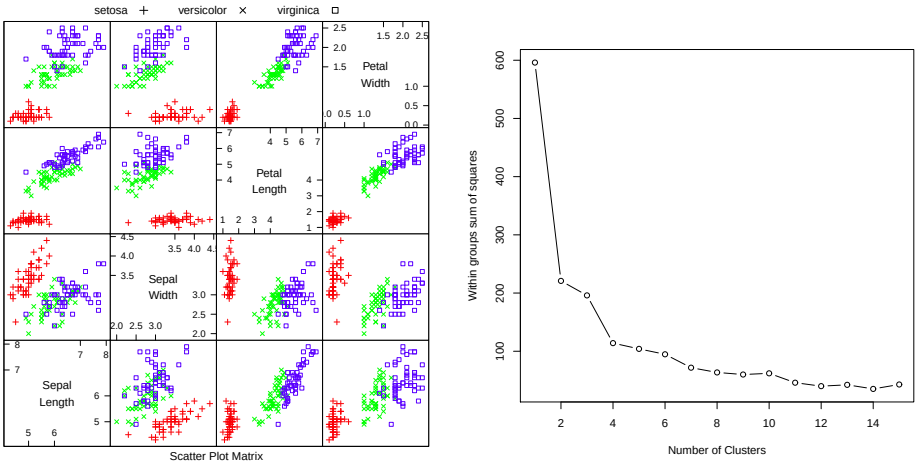


FIGURE 10.9: *On the* **left**, *the scatterplot matrix for the Iris data, for reference. On the* **right**, *a plot of the value of the cost function for each of several different values of* $k$. *Notice how there is a sharp drop in cost going from* $k = 1$ *to* $k = 2$, *and again at* $k = 4$; *after that, the cost falls off slowly. This suggests using* $k = 2$, $k = 3$, *or* $k = 4$, *depending on the precise application.*

each cluster by finding the medoid that minimizes the sum of distances of points in the cluster to that medoid (blank search is fine).

## 10.4 DESCRIBING REPETITION WITH VECTOR QUANTIZATION

Classifying images is an important problem, with a variety of applications. I list some below. We have seen powerful classification methods. But what features should one use to classify images? In this section, I will describe a construction that extends to a variety of other signals, because it exploits repetition. Repetition is an important feature of many interesting signals. For example, images contain *textures*, which are orderly patterns that look like large numbers of small structures that are repeated. Examples include the spots of animals such as leopards or cheetahs; the stripes of animals such as tigers or zebras; the patterns on bark, wood, and skin. Similarly, speech signals contain *phonemes* — characteristic, stylised sounds that people assemble together to produce speech (for example, the "ka" sound followed by the "tuh" sound leading to "cat"). An important part of representing signals that repeat is identifying the components that repeat, then describing the signal in terms of those components.

Repetition occurs in more subtle forms than spot patterns. The essence is that a small number of local patterns can be used to represent a large number of examples. You see this effect in pictures of scenes. If you collect many pictures of, say, a beach scene, you will expect most to contain some waves, some sky, and some sand. The individual patches of wave, sky or sand can be surprisingly similar, and different images are made by selecting some patches from a vocabulary of patches, then placing them down to form an image. Similarly, pictures of living

Choose $k$ data points to act as cluster centers
Until the cluster centers change very little
    First, we estimate the weights

    For $i$ indexing data points
        For $j$ indexing cluster centers
            Compute $s_{i,j} = e^{\frac{-\|\mathbf{x}_i - \mathbf{c}_j\|}{2\sigma^2}}$

    For $i$ indexing data points
        For $j$ indexing cluster centers
            Compute $w_{i,j} = s_{i,j} / \sum_{l=1} k s_{i,l}$

    Now, we re-estimate the centers

    For $j$ indexing cluster centers
        Compute $\mathbf{c}_j = \frac{\sum_i w_{i,j}\mathbf{x}_i}{\sum_i w_{i,j}}$

end

**Algorithm 10.4:** *Soft Clustering by K-Means.*

rooms contain chair patches, TV patches, and carpet patches. Many different living rooms can be made from small vocabularies of patches; but you won't often see wave patches in living rooms, or carpet patches in beach scenes.

This view suggests that we should (a) develop a vocabulary of image patches and then (b) construct a feature that describes which vocabulary elements appear in the image. It turns out that where the vocabulary elements appear is less important. This is because different pictures of a living room may have the same patches in different locations (for example, you might just move the camera to the left). As another example, zebras have stripes, but precisely where the stripes lie on a zebra doesn't seem to matter much in practice.

The patches in the vocabulary are often referred to as **textons**. We could find these textons by looking for image patches that are common, and then reason about which ones are repeated. There are two important difficulties in finding image patches that commonly occur together. First, these representations of the image are continuous. We cannot simply count how many times a particular pattern occurs, because each vector is slightly different. Second, the representation is high dimensional in either case. A patch around a pixel might need hundreds of pixels to represent it well. This means we cannot build a histogram directly, because it will have an unmanageable number of cells.

### 10.4.1 Applications of Image Classification

**Detecting explicit images:** There are numerous reasons to try and detect images

FIGURE 10.10: *Material is not the same as object category (the three cars on the* **top** *are each made of different materials), and is not the same as texture (the three checkered objects on the* **bottom** *are made of different materials). Knowing the material that makes up an object gives us a useful description, somewhat distinct from its identity and its texture.* This figure was originally published as Figures 2 and 3 of "Exploring Features in a Bayesian Framework for Material Recognition," by C. Liu, L. Sharan, E. Adelson, and R. Rosenholtz Proc. CVPR 2010, 2010 © IEEE, 2010.

that depict nudity or sexual content. In some jurisdictions, possession or distribution of such pictures might be illegal. Many employers want to ensure that work computers are not used to collect or view such images; in fact, vendors of image filtering software have tried to persuade employers that they might face litigation if they don't do so. Advertisers want to be careful that their ads appear only next to images that will not distress their customers. Web search companies would like to allow users to avoid seeing images they find distressing.

**Material classification:** Imagine we have an image window. What material (e.g., "wood," "glass," "rubber," and "leather") does the window cover? If we could answer this question, we could decide whether an image patch was cloth—and so might be part of a person, or of furniture—or grass, or trees, or sky. Generally, different materials produce different image textures, so natural features to use for this multiclass classification problem will be texture features. However, materials tend to have some surface relief (for example, the little pores on the surface of an orange; the bumps on stucco; the grooves in tree bark), and these generate quite complex shading behavior. Changes in the illumination direction can cause the shadows attached to the relief to change quite sharply, and so the overall texture changes. Furthermore, as Figure 10.10 illustrates, texture features might indicate the material an object is made of, but objects can have the same texture and be made of quite different materials.

**Scene classification:** Pictures of a bedroom, of a kitchen, or of a beach show different **scenes**. Scenes provide an important source of context to use in

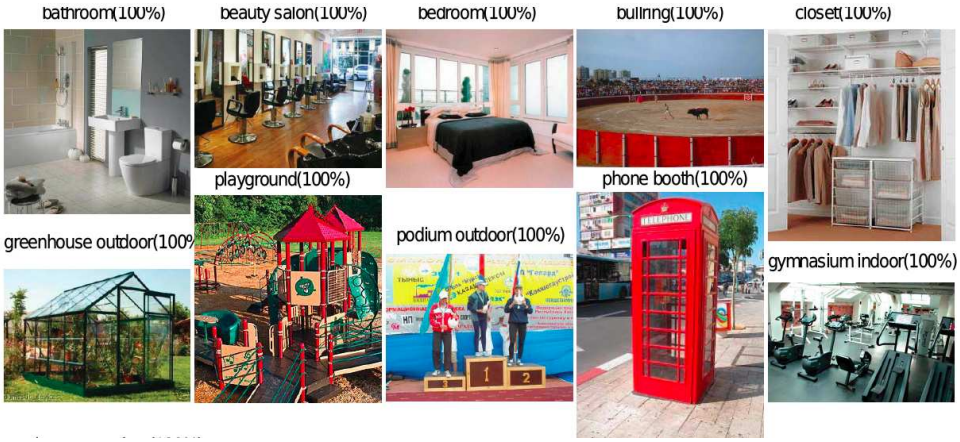


FIGURE 10.11: *Some scenes are easily identified by humans. These are examples from the SUN dataset (?) of scene categories that people identify accurately from images; the label above each image gives its scene type.* This figure was originally published as Figure 2 of "SUN database: Large-scale Scene Recognition from Abbey to Zoo," by J. Xiao, J. Hays, K. Ehinger, A. Oliva, and A. Torralba, Proc. IEEE CVPR 2010, © IEEE, 2010.

interpreting images. You could reasonably expect to see a pillow, but not a toaster or a beachball, in a bedroom; a toaster, but not a pillow or a beachball, in a kitchen; or a beachball and perhaps a pillow, but not a toaster, on a beach. We should like to be able to tell what scene is depicted in an image. This is difficult, because scenes vary quite widely in appearance, and this variation has a strong spatial component. For example, the toaster could be in many different locations in the kitchen. In **scene classification**, one must identify the scene depicted in the image. Scene labels are more arbitrary than object labels, because there is no clear consensus on what the different labels should be. Some labels seem fairly clear and are easy to assign accurately (Figure 10.11), for example, "kitchen," "bedroom," and other names of rooms in a house. Other labels are uncertain: should one distinguish between "woodland paths" and "meadows", or just label them all "outdoors"? Humans seem to have a problem here, too (Figure **??**). However, there are several scene datasets, each with its own set of labels, so methods can be compared and evaluated.

### 10.4.2 Vector Quantization and Textons

**Vector quantization** is a strategy to deal with these difficulties. Vector quantization is a way of representing vectors in a continuous space with numbers from a set of fixed size. We first build a set of clusters out of a training set of vectors; this set of clusters is often thought of as a dictionary. We now replace any new vector with the cluster center closest to that vector. This strategy applies to vectors quite generally, though we will use it for texture representation. Many different clusterers can be used for vector quantization, but it is most common to use k-means or one

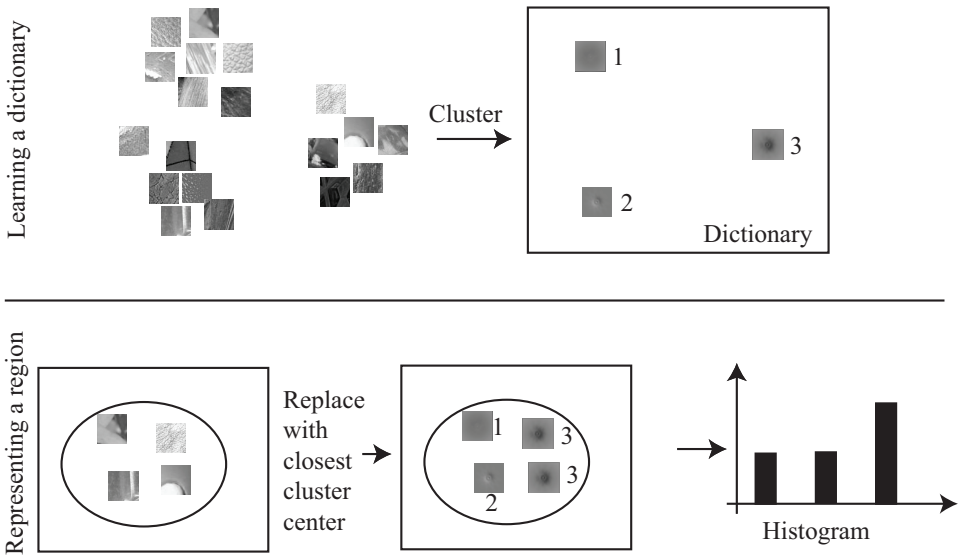


FIGURE 10.12: *There are two steps to building a pooled texture representation for a texture in an image domain. First, one builds a dictionary representing the range of possible pattern elements, using a large number of texture patches. This is usually done in advance, using a training data set of some form. Second, one takes the patches inside the domain, vector quantizes them by identifying the number of the closest cluster center, then computes a histogram of the different cluster center numbers that occur within a region. This histogram might appear to contain no spatial information, but this is a misperception. Some frequent elements in the histogram are likely to be textons, but others describe common ways in which textons lie close to one another; this is a rough spatial cue.* This figure shows elements of a database collected by C. Liu, L. Sharan, E. Adelson, and R. Rosenholtz, and published at `http://people.csail.mit.edu/lavanya/research_sharan.html`. Figure by kind permission of the collectors.

of its variants.

We can now represent a collection of vectors as a histogram of cluster centers. This general recipe can be applied to image representation by describing each pixel in the domain with some vector, then vector quantizing and describing the domain with the histogram of cluster centers. One natural vector to use is obtained by reshaping the pixels from a fixed-size patch around the image pixel (Figure 10.13). There is a rich collection of alternative possibilities in the computer vision literature.

### 10.4.3 Hierarchical K Means

One important difficulty occurs in applications. We might need to have an enormous dataset (millions of image patches are a real possibility), and so a very large $k$. In this case, $k$ means clustering becomes difficult because identifying which cluster center is closest to a particular data point scales linearly with $k$ (and we have to

Build a dictionary:
    Collect many training example images
    Construct the vectors $\mathbf{x}$ for relevant pixels; these could be
      a reshaping of a patch around the pixel, or a vector of filter outputs
      computed at the pixel, or other image representations.
    Obtain $k$ cluster centers $\mathbf{c}$ for these examples

Represent an image domain:
    For each relevant pixel $i$ in the image
        Compute the vector representation $\mathbf{x}_i$ of that pixel
        Obtain $j$, the index of the cluster center $\mathbf{c}_j$ closest to that pixel
        Insert $j$ into a histogram for that domain

**Algorithm 10.5:** *Image Representation Using Vector Quantization.*

do this for every data point at every iteration). There are two useful strategies for dealing with this problem.

The first is to notice that, if we can be reasonably confident that each cluster contains many data points, some of the data is redundant. We could randomly subsample the data, cluster that, then keep the cluster centers. This works, but doesn't scale particularly well.

A more effective strategy is to build a hierarchy of k-means clusters. We randomly subsample the data (typically, quite aggressively), then cluster this with a small value of $k$. Each data item is then allocated to the closest cluster center, and the data in each cluster is clustered again with k-means. We now have something that looks like a two-level tree of clusters. Of course, this process can be repeated to produce a multi-level tree of clusters. It is easy to use this tree to vector quantize a query data item. We vector quantize at the first level. Doing so chooses a branch of the tree, and we pass the data item to this branch. It is either a leaf, in which case we report the number of the leaf, or it is a set of clusters, in which case we vector quantize, and pass the data item down. This procedure is efficient both when one clusters and at run time.

### 10.4.4 Using Textons

We can now build a large vocabulary of textons. Details vary from application to application, but the general procedure is clear. We obtain a large number of relevant images (from thousands to millions, depending on the particular application). We extract patches from these images. These patches could have centers that are on a grid, or are randomly chosen, or lie at image corners (which are detected using methods outside the scope of this discussion). These patches are clustered using k-means or hierarchical k-means. It is quite usual to have $k$ in the hundreds of thousands. We can then describe a new image by decomposing it into patches (typically, on a grid); vector quantizing each patch; and forming a histogram of the centers. Usually, this histogram is normalized, so that larger images (with more

patches) do not appear different from smaller images.

When we use this description, we need to keep in mind that it is a histogram. Histograms can be represented as vectors (one entry per bin), but in important respects they are not really vectors. All the entries in a normalized histogram sum to one. This means that measuring the difference between two histograms with a Euclidean distance measure is not a particularly good idea. The significance of a large difference at a particular bin depends on how full the bin is. Write $\mathbf{g}$ and $\mathbf{h}$ for the two histograms we wish to compare, and $g_i$ (resp. $h_i$) for the $i$'th bin. Now assume $(h_i - g_i)^2$ is large. If $h_i + g_i$ is large, too, this might not be significant, because we might have a difference in counts that is due to random effects. But if $h_i + g_i$ is small, the difference is much more significant, because one bin has many entries and another has few. This argues for using a distance

$$d(\mathbf{h}, \mathbf{g}) = \sum_i \frac{(h_i - g_i)^2}{(h_i + g_i)}$$

which is known as the $\chi^2$ distance (often written as the chi-squared distance). The name comes from an analogy with Pearson's $\chi^2$ distance for comparing histograms (which is $\sum_i \frac{(h_i - g_i)^2}{h_i}$).

We can now use this image description in a variety of ways.

- **Image classification with SVM's:** One could use an SVM to classify images. Better results are available with more complex classifier technologies, because SVM's do not account for the histogram distance effect, above.

- **Image classification with nearest neighbors:** A nearest neighbor classifier should use the chi-squared distance. It is often sufficient to use an approximate nearest neighbor method to get a set of possible histograms close to the query, then find the closest using the chi-squared distance.

- **Image retrieval:** There are many applications where we would like to recover images that are similar in appearance to a query image. We can do so by selecting images that have a small chi-squared distance to the query image. Some complex technical paraphenalia are required to make doing so efficient, however.

- **Image clustering:** We can cluster images using the chi-squared distance.

One issue that arises quite regularly in applications is that vector quantization loses information. Small changes in image patch can lead to a large change in the cluster center, and many of these can accumulate to give a big deviation in the histogram. This is an important source of error in applications. There is an effective strategy to handle it. One builds multiple systems, then votes. For example, imagine building a nearest neighbor classifier for images. We build (say) three, rather than one, vector quantizers. Each clusters the same set of image patches, but uses a different random start. We then build three (say) rather than one histogram describing an image, and query three collections, one with each histogram. Finally, we choose the category that gets the majority of votes.

An alternate use of a texton vocabulary is in denoising. Assume we have a large vocabulary of image patches, built as above. A new image comes in, but is noisy, perhaps because there are errors in the storage and retrieval system, or in a communication system, or because the camera is very slightly defocussed. We can build a reconstruction by matching each of the noisy image patches to its cluster center, then replacing the patch with the cluster center. The details are complex, because we need to figure out what to do with patches that overlap, and to ensure that patches that abut one another are consistent.

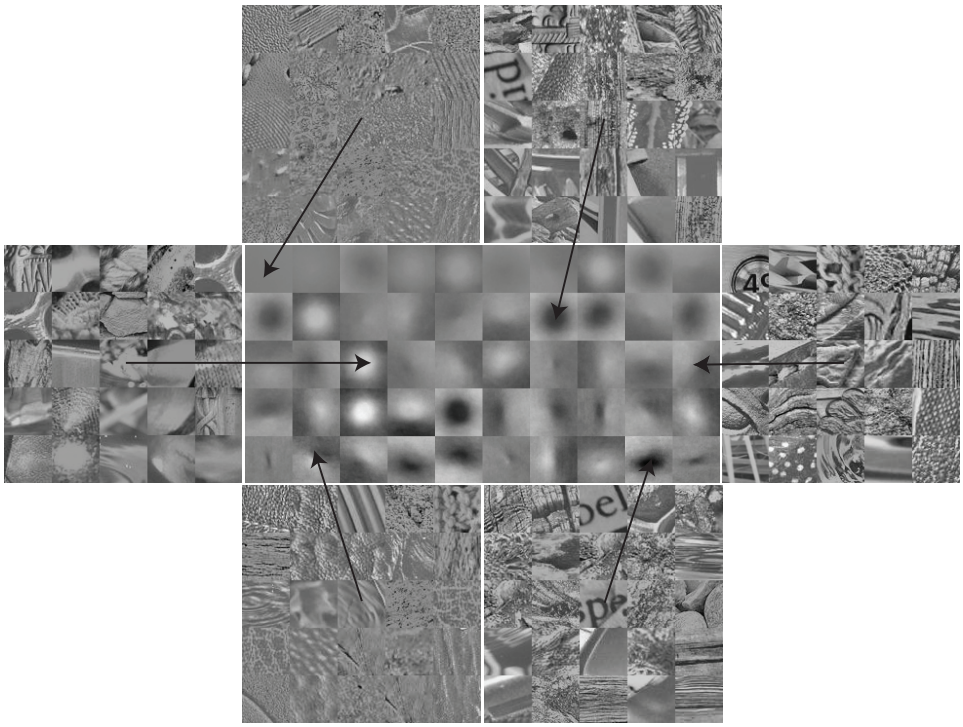

FIGURE 10.13: *Pattern elements can also be identified by vector quantizing vectors obtained by reshaping an image window centered on each pixel. Here we show the top 50 pattern elements (or textons), obtained using this strategy from all 1,000 images of the collection of material images described in Figure* **??**. *Each subimage here illustrates a cluster center. For some cluster centers, we show the closest 25 image patches. To measure distance, we first subtracted the average image intensity, and we weighted by a Gaussian to ensure that pixels close to the center of the patch were weighted higher than those far from the center.* This figure shows elements of a database collected by C. Liu, L. Sharan, E. Adelson, and R. Rosenholtz, and published at `http://people.csail.mit.edu/lavanya/research_sharan.html`. Figure by kind permission of the collectors.