Class Notes for B365: Data Analysis and Mining

Christopher Raphael

March 28, 2023

Chapter 1

Preliminaries

1.1 Data

The general scenario we will treat in this class involves data. Our data will come from a "population" which is the world we are trying to understand. This could be a literal population, like the people who live in a particular country or state, though it applies just as well to any collection of objects, people, animals, transactions, or anything else we hope to understand through data analysis. For instance, our population could be the deer of southern Indiana, the summers of the 19th century, the postcards mailed in the last month, or really any possible category of things.

We come to understand our population by "sampling" from the population, meaning observing members (not necessarily people) while measuring certain aspects of the members. Polling would be a familiar example of this sampling process. There is a variety of different terminology that is used to discuss this sampling scenario. We will call the members we sample the *observations* or *instances*. On each instance we typically measure several (maybe many) aspects of the instance. These measurements are called *features* or *variables*.

For instance, suppose we sample a collection of students at a particular university and measures 4 variables: the hours/week a the student works at a job; the hours/week the student studies; the hours/week the student spends on extracurricular activities, and the student's GPA. We could present these data in a *data matrix* as follows:

	job	study	extracurricular	GPA
Student 1	10	10	0	3.2
Student 2	5	15	10	3.8
Student 3	0	5	0	2.0
⋮	⋮	⋮	⋮	⋮

The *rows* of the above matrix are the observations (or instances); the columns of the matrix are the features (or variables). We will use this matrix with rows and columns interpreted as above for the remainder of the class.

In the example above all of the features or variables are numbers, but this is not always the case. Common types of variables are listed below.

Binary variables *Binary* variables have two possible values such as yes/no, 0/1, accept/reject, survived/not, like/dislike, etc.. It will be common to "code" these outcomes as 1/0 in our data matrix, even if the interpretation is not numeric.

Categorical A *categorical* variable takes on one of several different possibles types. For instance

- A blood type (ignoring the Rh factor) could be O, A, B, or AB.
- A US state could be AL, AK, AZ, . . . , WY.
- A cloud could be cirrus, cumulus, stratus, or nimbus.
- A citrus fruit could be orange, lemon, lime, grapefruit, kumquat, other.

While there are simply 50 states, one could make finer or coarser categorizations of clouds, blood types or citrus. This will often be the case with categorical variables, though the choice of granularity is made when the data are collected, so there isn't much we can do about this.

Ordinal An *ordinal* variable is a categorical variable where the categories are ordered. A common example is the "Likert" scale where a person is given a statement and asked if they: strongly agree, agree, are neutral, disagree, or strongly disagree. Another example would be the number of cylinders in a car, say 4,6, or 8; or a rating of pain on a scale of 1 to 10. An ordinal variable gives us a little more detailed information than a non-ordinal categorical variable. For instance it would be reasonable to group the people who either agreed or strongly agreed as having a similar response, though non-ordinal categorical variables provide no such guide to coarsening their granularity.

Continuous A *continuous* variable has a real number value like 3.63. Examples of continuous variables are weight, length, lifetime, temperature, interest rate, blood pressure, etc. In general, continuous variables are the most flexible types of variables since we can always make binary, or ordinal variables out of them by "binning." For instance, a binary variable can be created by asking whether a continuous variable is greater than or less than 0. Or an ordinal variables could be created by asking if the continuous variables is less than 0, between 0 and 10, or greater than 10. It isn't possible to create a true continuous variable from a non-continuous variable.

It is worth mentioning that our data matrix may not be completely filled with values. Sometimes certain values are missing, perhaps because they were never measured, or lost at some stage of the process. Typically these *missing values* are represented in the data matrix as NA, for "not available", though other codings are possible. It is often possible to still reason effectively with data containing missing values, though we will not treat this topic in our course.

People usually collect data because they have an idea how it may be useful. Often the measured variables are of two kinds: *predictor* and *response* where we want to use the predictor variables to estimate what the outcome of a response variable will be.

For instance consider the situation of the reedmaker. Many musical instruments, such as the clarinet, saxophone, and bagpipes make sound with a vibrating reed. The reed is made from a bamboo-like plant, officially called *Arundo Donax*, though informally called "reed cane," through a painstaking process that requires a significant investment in each reed as it is adjusted and played in an attempt to make it work properly. In fact, sometimes the reed is completely rejected after it fails the process, resulting in lost time for the reedmaker. Some reedmakers believe that some pieces of cane are more likely to succeed than others, and hope to determine this *before* much time is invested.

In this case the predictor variables are aspects of the cane that are measured before the reed is made. These may include:

Hardness How hard is the piece of cane? This could be measured twisting the cane when it is thoroughly soaked and seeing how much force is required to make it bend. Suppose this is measured on an *ordinal* scale of soft, medium, hard.

Graininess The cane is fibrous and looks something like wood, though the grain is more regular. Sometimes the fibers are large (coarse-grained) though sometimes they are smaller (fine-grained). Again we measure this on an ordinal scale of coarse, medium, fine.

Sink Time When the dry cane is place in water it does not sink right away, though eventually does as it absorbs water. The measurement of sink time, in minutes, is a continuous variable.

Result After the reed is made we measure the quality of the result. This is our *response* variable and is the thing we care most about. Here we measure quality on a continuous scale of 1 to 5, 1 being the worst and 5 being the best.

After we collect our data we would get a data matrix as follows:

	Hardness	Grain	Sink time	Result
Reed 1	soft	fine	20	4
Reed 2	medium	medium	10	3.5
Reed 3	hard	coarse	30	1
⋮	⋮	⋮	⋮	⋮

The goal here would be to try to predict the result from the variables we measure. If we could do this with only partial success we could still save a lot of time in the reedmaking process. It is likely that you can think of many different procedures, like reedmaking, that might be improved by using similar data analysis to guide the process.

1.2 The R Programming Environment

We will use the R environment for the computing we do in this class including both homework and in-class examples. R works well for us because it is possible to do a lot with only a few lines of code, while the resulting code is transparent and easy-to-understand. R is a high-level language, meaning that much is "taken care of" for the programmer — for instance we don't usually worry about the types of variables such as "int" and "float" since the language handles that for us. R is a vector-oriented language, meaning that one can process collections of numbers (vectors) with single commands. Thus you don't need to write many loops in R. Perhaps the best thing about R is how easy it is to get answers to your programming questions. The program is widely used, so internet searches usually produce helpful results on a wide variety of topics. Finally, R is great for data visualization (plotting data), which is an integral topic of our class.

The Canvas page has a tutorial introduction to R. We will work through this in class, but you should still do this by yourself since you will understand it and remember it better this way.

1.3 Probability Introduction

Probability deals with experiments where the outcome is viewed as random — happening according to chance rather than through the application of some reliable principles. If we had a machine the flipped a coin by tossing it into the air, it is possible that the outcome is not random. If we knew the exact position where the coin would be struck, thus knowing exactly how fast it would spin, and exactly how high it would be thrown into the air, and exactly the rate the spinning would slow down due to air resistance, etc., we might be able to figure out which side of the coin would land face up. But it sure is a lot easier to model the experiment as a probabilistic one: heads and tails are equally likely to occur and the result is *random*.

In our course we will deal with experiments having a finite number of possible outcomes $\omega_1, \omega_2, \ldots \omega_n$. The *sample space*, Ω, is the set of possible outcomes

$$\Omega = \{\omega_1, \omega_2, \ldots, \omega_n\}$$

For example, if we flip a coin then $\Omega = \{H, T\}$ or if we roll a die then $\Omega = \{1, 2, 3, 4, 5, 6\}$, or if we choose an M&M from an (original) bag and observe its color then $\Omega = \{\mathsf{red}, \mathsf{yellow}, \mathsf{violet}, \mathsf{green}, \mathsf{brown}\}$ It would be reasonable to think that both sides of the coin are equally likely to occur, with the same statement for the 6 faces of the die. But it wouldn't be true that the M&M colors are equally likely since the colors exist in different proportions.

Each possible outcome, ω_i has a *probability*, written $P(\omega_i)$, giving its likelihood of occurring. This probability is a number between 0 and 1 (inclusive) with the interpretation:

1. $P(\omega_i) = 0$ means ω_i is impossible.
2. $P(\omega_i) = 1$ means ω_i is certain to occur.
3. However, if $0 < P(\omega_i) < 1$ then $P(\omega_i)$ is the proportion of times that ω_i would occur if the experiment is repeated forever.

For example, in the coin flipping experiment we have $P(H) = P(T) = 1/2$ meaning that heads and tails would each come up half the time if we flipped the coin forever.

A basic assumption we make is that all of the probability must sum to 1:

$$P(\omega_1) + P(\omega_2) + \ldots P(\omega_n) = \sum_{i=1}^{n} P(\omega_i) = 1$$

Note: If you are unfamiliar with the sum notation with the "Σ," think of it as equivalent to a loop:

```
for (i=1,sum=0; i <= n; i++) sum += omega[i];
```

Example (Single Die) Roll a 6-sided die. Then $\Omega = \{1, 2, 3, 4, 5, 6\}$ with

$$P(1) = P(2) = P(3) = P(4) = P(5) = P(6) = \frac{1}{6}$$

Example (Blood Types) Blood types can be represented by a group: $\{A, B, O, AB\}$ and an RhD status: $\{+, -\}$ so

$$\begin{aligned} \Omega &= \{A, B, O, AB\} \times \{+, -\} \\ &= \{A+, A-, B+, B-, O+, O-, AB+, AB-\} \end{aligned}$$

(recall that the $\{A, B, O, AB\} \times \{+, -\}$ notation means all ways of creating "ordered pairs" from the two sets. That is, all ways of taking one from the first set and one from the 2nd set.) In a particular population the proportions may look something like what is below — the probabilities describe the likelihoods of the possible blood types if we were to draw an individual at random from the population.

Type	Percentage	Probability
O^+	38%	.38
O^-	7%	.07
A^+	34%	.34
A^-	6%	.06
B^+	9%	.09
B^-	2%	.02
AB^+	3%	.03
AB^-	1%	.01

Example (Pair of dice) Roll a pair of dice.

$$\begin{aligned} \Omega &= \{1, 2, 3, 4, 5, 6\} \times \{1, 2, 3, 4, 5, 6\} \\ &= \{(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (2, 1), \ldots, (6, 6)\} \end{aligned}$$

Note that there are 36 possible pairs: ($|\Omega| = 36$). Since all elements of Ω are equally likely

$$P(1, 1) = P(1, 2) = P(1, 3) = \ldots = P(6, 6) = \frac{1}{36}$$

Events An *event*, A, is a subset of the sample space: $A \subseteq \Omega$. You can think of an event as something that can happen with the random experiment. In particular, an event, A, *occurs* if the outcome, ω_i, of the random experiment is in A: $\omega_i \in A$. The probability of A is the sum of all the probabilities that compose A:

$$P(A) = \sum_{\omega_i \in A} P(\omega_i)$$

Example For the pair of dice let A be the event that the dice sum to 4. Thus

$$A = \{(1, 3), (2, 2), (3, 1)\} \subseteq \Omega$$

and

$$P(A) = P(1, 3) + P(2, 2) + P(3, 1) = \frac{1}{36} + \frac{1}{36} + \frac{1}{36} = \frac{1}{12}$$

Example For the blood types we can look at the event O that results when the the blood group is O with either Rh factor. Thus $O = \{O^+, O^-\}$ and

$$P(O) = P(O^+) + P(O^-) = .38 + .07 = .45$$

"Or", "And", and "Not" Since events are subsets, it is worth considering how the most basic operations on subsets relate to events.

Or First of all, if A and B are events then their union, $A \cup B$, is also an event. If $A \cup B$ occurs then the outcome must be in $A \cup B$, so the outcome must be in A *or* the outcome must be in B. That is, $A \cup B$ occurs if and only if A *or* B occurs. Union is like "or" in the sense that $P(A \text{ or } B) = P(A \cup B)$

And By similar reasoning, $A \cap B$ occurs if and only if A occurs *and* B occurs. Intersection is like "and" in the sense that $P(A \text{ and } B) = P(A \cap B)$

Not Finally, we will write A^c for the *complement* of A — all the elements of the sample space, Ω that are *not* in A. A^c occurs if and only if A does *not* occur. Complement is like "not" in the sense that $P(\text{not } A) = P(A^c)$

We will keep these interpretations in mind of "or," "and," and "not" going forward.

Mutually Exclusive Events Two events are said to be *mutually exclusive* if one occurring means the other cannot occur. For example, if we have a random experiment that produces a number, the number can be odd (O) or even (E). Note that both O and E are events since they correspond to subsets of the possible outcomes of the experiment. Since a number can't be both even and odd, then O and E are mutually exclusive. It is also possible to have several mutually exclusive events. For instance, a object selected from a box of *Cracker Jacks* could be either a piece of caramel corn, a peanut, or a prize, but it could not be *two* of these categories.

We can relate this to the language of sets, as follows. If events A and B are mutually exclusive, this is really just another way of saying that A and B are disjoint: $A \cap B = \emptyset$ ("$\emptyset$" is notation for the set with no elements). This is because A, B mutually exclusive means it is not possible to for both A and B to occur, meaning that there is no outcome that is in both A and B. That is $A \cap B = \emptyset$.

Axiomatic Description of Probability Sometimes probability is viewed in a more abstract way than we have done so far. We won't stress this more abstract or "axiomatic" view of probability in our class, but it is important enough to at least mention.

For P to be a legitimate probability it must satisfy the following axioms:

1. For all events, A, $P(A) \geq 0$
2. $P(\Omega) = 1$
3. For mutually exclusive events $A_1, A_2, \ldots, A_n$,

$$P(A_1 \cup A_2 \cup \ldots \cup A_n) = P(A_1) + P(A_2) + \ldots + P(A_n)$$

or, with more compact notation,

$$P(\bigcup_{i=1}^{n} A_i) = \sum_{i=1}^{n} P(A_i)$$

The first axiom states that probabilities are numbers that are 0 or greater. We have already used this when we said that the probability of an element of the sample space, $\omega_i \in \Omega$, had $P(\omega_i) \geq 0$.

We have also used the last two axioms as well when we stated that for $\Omega = \{\omega_1, \omega_2, \ldots, \omega_n\}$ we must have $P(\omega_1) + P(\omega_2) + \ldots + P(\omega_n) = 1$. Since the individual elements ω_i can be thought of as mutually exclusive (=disjoint) one-element subsets then, using the last two axioms, we get

$$1 = P(\Omega) = P(\bigcup_{i=1}^{n} \omega_i) = \sum_{i=1}^{n} P(\omega_i)$$

By similar reasoning we used these two axioms we defined the probability of an event, A, as $P(A) = \sum_{\omega_i \in A} P(\omega_i)$. So, we could have began with the axioms and got all the properties we need for probability.

Example: Rolling a 4,6, or 8 Roll a pair of dice and let

$$\begin{aligned} A &= \text{the sum of dice is 4} \\ B &= \text{the sum of dice is 6} \\ C &= \text{the sum of dice is 8} \end{aligned}$$

Viewing the events A, B, C as sets we have

$$\begin{aligned} A &= \{(1,3),(2,2),(3,1)\} \\ B &= \{(1,5),(2,4),(3,3),(4,2),(5,1)\} \\ C &= \{(2,6),(3,5),(4,4),(5,3),(6,2)\} \end{aligned}$$

so $P(A) = \frac{3}{36}$, $P(B) = \frac{5}{36}$, $P(C) = \frac{5}{36}$. Since A, B, C are mutually exclusive then

$$\begin{aligned} P(\text{dice sum is 4 or 6 or 8}) &= P(A \cup B \cup C) \\ &= P(A) + P(B) + P(C) \\ &= \frac{3}{36} + \frac{5}{36} + \frac{5}{36} \\ &= \frac{13}{36} \end{aligned}$$

Random Variables We often have experiments that result in numbers, such as our pair of dice experiment. It often creates simpler notation if we refer to the outcome as a *random variable*. Usually capital letters are used to denote random variables, so we may let X be the random variable that is the sum of the two dice. In this case we can write events such as $(X = 6) = \{(1,5),(2,4),(3,3),(4,2),(5,1)\}$ and write $P(X = 6)$ for its probability. This is the notation you will typically see in the world of probability. It looks a little different than what we have done before, but it really is nothing new. Furthermore, it is needed for clarity later when there are multiple random variables involved.

1.4 Simulating Probabilities

In some cases we may not know how to do the calculation of a particular probability. Sometimes there is no way to do such a calculation. For example suppose we flip a coin 10 times and write Y for the number of heads we get. It turns out that probability of getting exactly 5 heads, $P(Y = 5)$ is given by

$$P(Y = 5) = \frac{10!}{(5!)(5!)(2^{10})} = .2460938\ldots$$

where $n!$ (read as "n *factorial*") has $n! = n \times (n-1) \times (n-2) \times \ldots \times 1$. *But what if we didn't know this?*

Here we can leverage our basic intuitive understanding of a probability as the proportion of times a certain event will occurs if repeated forever. Of course, we can't repeat an experiment forever, but we can do it enough times to *approximate* the probability. The computer will be useful in doing this. In general, we would expect a better and better approximation as we simulate the experiment more and more times.

R Example: simulate_single_coin_flip.r

R Example: simulate_multiple_coin_flips.r

R Practice Example: Prob of rolling 8 with pair of dice

How Accurate is Probability Estimate from Simulation? We want to know $P(A)$ for some event A. We simulate the experiment n times and suppose that $\#A$ is the number of times A occurs. We will write $\hat{p}$ for our estimate of $P(A)$ from this experiment. The obvious estimate is

$$\hat{p} = \frac{\#A}{n} = \text{ the proportion of times } A \text{ occurred}$$

But this estimate is not very useful if we don't have some idea how accurate it is, and, of course, this will depend on n. Here is a simple "rule of thumb" that describes the accuracy of $\hat{p}$. We will refine this later with a more precise rule, though the rule of thumb is easier to remember and gives the right idea.

The $\frac{1}{\sqrt{n}}$ Rule The interval

$$\hat{p} \pm \frac{1}{\sqrt{n}} = (\hat{p} - \frac{1}{\sqrt{n}}, \hat{p} + \frac{1}{\sqrt{n}})$$

will contain the true probability, $P(A)$, at least 95% of the time. Said another way, our error, e, in estimating the true probability is $e = |\hat{p} - P(A)|$ Then

$$e < \frac{1}{\sqrt{n}}$$

with probability at least .95. In statistical terminology this can be stated as: $\hat{p} \pm \frac{1}{\sqrt{n}}$ is a 95% *confidence interval* for $P(A)$.

A couple of R examples help illustrate this idea through simulation:

1. single_flip_revisited.r
2. multiple_flips_revisited.r

In these examples, rather than getting a single number for a probability estimate, we get a 95% confidence interval. We see that if perform the simulations over and over, thus getting a whole collection of confidence intervals, about 95% of the confidence intervals contain the probability we are trying to estimate. This makes is clear what the "95" in the 95% confidence interval means.

Example: Polling We want to know the proportion of voters that favor a particular candidate A. If we choose a single voter at random, the probability that this voter favors candidate A, $P(A)$, is the same as the proportion of voters favoring A. We sample n voters at random and let $\hat{p}$ be the fraction of the *sampled* voters that favor A. Then $\hat{p} \pm \frac{1}{\sqrt{n}}$ contains $P(A)$ with probability at least .95. This is usually stated as: "This poll carries a margin of error of m" where $m = \frac{1}{\sqrt{n}}$.

Given that the mathematics for polling accuracy is so simple, you may wonder why there have been so many cases in history where polls have been inaccurate. One reason, of course, is that the things we try to measure, like the proportion of voters favoring a candidate, are not a fixed numbers, but rather things that change over time. But the more important reason is the near impossibility of sampling a population at random. The first thing you might try is selecting some members of the population at random and asking your question. The problem with this is that you will get answers from some and not others. There are many possible reasons for this, such as the availability of the individual, or their willingness to give their time to your survey. Whatever the reasons, the actual sample you obtain is *not* from the population of interest, but, rather, from the sub-population you could get answers from. This population is different from your population of interest. For example, perhaps you managed to get answers from people who had more time to answer questions, and thus tended to be older. This survey-answering group will have a different proportion favoring the candidate of interest, thus skewing your results. It is very difficult to find a way of sampling a population that doesn't implicitly favor some members over others.

Confidence Interval Refinement The $\frac{1}{\sqrt{n}}$ rule is easy to remember and states the big picture accurately. Given that the (half) width of the confidence interval is $\frac{1}{\sqrt{n}}$, if you want to make the confidence interval half as big you will need *4 times as many samples.* This is because $\sqrt{4n} = 2\sqrt{n}$.

A more precise formula for the 95% confidence interval is given now. If we want to estimate some probability or proportion, p, and randomly sample n members of the population to get $\hat{p}$, then a better 95% confidence interval is given by

$$\hat{p} \pm 1.96\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$$

If you consider this interval you'll see that the width still has a $\sqrt{n}$ in the denominator, so it behaves similarly to what we had before. In particular, we still need 4 times as many samples to make the interval twice as small. But it will will produce smaller intervals when the true probability is near 0 or 1. Smaller intervals are a good thing, since they are more definite in describing the possible range for our desired probability. This is what we will use in the remainder of our class.

Example: Refined Confidence Interval A machine produces a long run of parts, some of which succeed and some of which fail. We sample 258 of these parts at random and find that 87 of these fail. Assuming that the failure rate is constant, what can be said about the failure rate?

$$\begin{aligned} P(F) &= \text{the true failure rate} \\ \hat{p} &= \frac{87}{258} \approx .34 \end{aligned}$$

A 95% confidence interval for the failure rate is given by

$$\begin{aligned} \hat{p} \pm 1.96\sqrt{\frac{\hat{p}(1-\hat{p})}{n}} &= .34 \pm 1.96\sqrt{\frac{(.34)(.66)}{258}} \\ &= .34 \pm .06 \end{aligned}$$

That is, we are 95% sure that the true failure rate lies between .34 -.06 = .28 and .34 + .06 = .40.

Example: Reasoning with Confidence Intervals (Statistical Significance) Confidence intervals add some sophistication to the way we interpret survey results or other experimental data. Let's consider now an example where we interpret the results both with and without using confidence intervals to see their value.

In our example, 300 people take a concentration test before and after a strongly caffeinated cup of coffee. 165 of these subjects have improved scores after the "treatment." *Does caffiene help concentration?*

If we assume that you can't get exactly the same score on the test twice, then, even if caffeine has no effect still 1/2 of the population would get a better score after the coffee, just due to random variation. So if we let $P(I)$ be the proportion of the population who get a better score after caffeine then we have a special interpretation of the case where $P(I) = .5$:

$$P(I) = .5 \Longleftrightarrow \text{Caffeine has no effect on concentration}$$

Here we use the symbol $\Longleftrightarrow$ to mean "is the same as." What follows is two possible analyses of our data.

1. We see that $\hat{p} = \frac{165}{300} = .55 > .5$ so we may conclude that people do better after caffeine.

   But .55 is not much bigger than .5. Is this difference enough to justify a conclusion?

2. A 95% confidence interval for $P(I)$ is

$$\begin{aligned} \hat{p} \pm 1.96\sqrt{\frac{\hat{p}(1-\hat{p})}{n}} &= .55 \pm 1.96\sqrt{\frac{(.55)(.45)}{300}} \\ &= .55 \pm .06 \end{aligned}$$

   Note that the special value of .5 (no effect) lies *inside* the confidence interval, so .5 (no effect) is a reasonable value for $P(I)$. *Do not conclude that caffeine helps concentration.*

How would you reason if, instead, there are 185 out of 300 "subjects" whose concentration improved?

1.5 Probability with Two Random Variables

We motivate the notion of two, or several, random variables by revisiting an example we have already considered, the blood type. This new view will allow us to ask some questions that can't easily be posed in our original view. We now view the earlier blood type probability table in terms of two components, the blood group, which we will put in the rows of a table, and the Rh factor, which we will put in the columns, as below.

		Rh factor	
		+	-
Group	A	.34	.06
	B	.09	.02
	AB	.03	.01
	O	.38	.07

Simply presenting the probabilities in this tabular form makes some things apparent that were buried in our original presentation, such as the greater probability of a positive Rh factor in each group. As notation we can write G as the blood group random variable and R for the Rh factor variable. If we were to sample a person randomly from the popluation we write, for example

$$P(G = A, R = +) = P(G = A \text{ and } R = +) = .34$$

for the event that the person's blood group is A *and* their Rh factor is +; that is, their blood type is A^+. The usual convention is to write the "and" as a comma, rather than using the word "and." One could also write, for instance,

$$\begin{aligned} P(G = O, R = -) &= .07 \\ P(G = AB, R = +) &= .03 \\ P(G = A, R = -) &= .06 \\ &\vdots \end{aligned}$$

for other statements reflected in the table. Since all possible combinations of the two variables G and R are presented, this is known as the *joint distribution* of the two variables.

Marginal Probability Suppose we consider the event $(G = O)$ — the blood group is type O. This includes *either* Rh factor. If we were to enumerate the possible ways this event could occur we see that

$$(G = O) = (G = O, R = +) \cup (G = O, R = -)$$

(remember that the comma means "and"). Thus the event $(G = O)$ has been written as a union of disjoint events so we can get the probability of $(G = O)$ as

$$\begin{aligned} P(G = O) &= P(G = O, R = +) + P(G = O, R = -) \\ &= .38 + .07 \\ &= .45 \end{aligned}$$

or by similar reasoning

$$\begin{aligned} P(R = +) &= P(G = A, R = +) + P(G = B, R = +) + P(G = AB, R = +) + P(G = O, R = +) \\ &= .34 + .09 + .03 + .38 \\ &= .84 \end{aligned}$$

It is common to include such probabilities in margin of the table, as below, by taking row sums and column sums:

		Rh factor		
		+	-	
Group	A	.34	.06	.40
	B	.09	.02	.11
	AB	.03	.01	.04
	O	.38	.07	.45
		.84	.16	1.

It is common to call this last column of the table the *marginal distribution* of G and the last row of the table the *marginal distribution* of R. Note that these two marginal distributions sum to 1 as probability distributions should. The marginal distributions reflect what we would observe if we measured *only* the blood group or *only* the Rh factor. It is worth noting that when we observe data in the real world we are always observing some kind of marginal distribution since there are many things that could be measured on any individual, though we will not measure them all. We will come back to this idea later.

Conditional Probability From the table above we see that that the probability of observing a positive Rh factor is $P(R = +) = .84$. This is the probability of positive Rh when we sample from the entire population. But what if we were only sampling from blood type O? Would the probability of getting a positive Rh factor be the same in this group? If we restrict our attention to blood group O, then we are simply looking at the row of the table corresponding to O. These two numbers, .38 and .07 don't make sense as probabilities since they don't sum to 1, but we could scale them so that they do:

$$\frac{.38}{.38 + .07} + \frac{.07}{.38 + .07} = 1$$

These would be the proportions of the two Rh types in the O group, so these two numbers are the probabilities of Rh types + and - *restricting our attention to blood group O.* These kinds of restricted probabilities are known as *conditional probabilities.* For instance, the probability of getting a positive Rh type *given* that the blood group is O is written as

$$P(R = +|G = O) = \frac{.38}{.38 + .07} = .84$$

The vertical bar, |, is read "given." Similarly, restricting our attention to subjects with positive Rh factor we could say

$$P(G = O|R = +) = \frac{.38}{.34 + .09 + .03 + .38} = .45$$

Summary and Generalization More generally, suppose we have two random variables X, Y. We will write $P(X = x, Y = y)$ for the probability that X takes on the value x *and* Y takes on the value y. That is, $P(X = x, Y = y)$ is the joint distribution of X and Y. We get the marginal distributions for X and Y as

$$\begin{aligned} P(X = x) &= \sum_y P(X = x, Y = y) \\ P(Y = y) &= \sum_x P(X = x, Y = y) \end{aligned}$$

where the sum $\sum_x$ is taken over all possible values of x and $\sum_y$ is taken over all possible values of y. The conditional distributions are given by

$$\begin{aligned} P(X = x|Y = y) &= \frac{P(X = x, Y = y)}{P(Y = y)} \\ P(Y = y|X = x) &= \frac{P(X = x, Y = y)}{P(X = x)} \end{aligned}$$

Working with a Sample Typically we do not know the exact distribution of the variables we consider, but rather, we have a sample of their outcomes. For example consider the following sample from a population of people who were trying to quit smoking. Some of these people were given a medicated path while others were given an unmedicated patch. The subjects are assigned randomly and don't know which kind of patch they had. After wearing their patch, some of the people successfully abstained from smoking, while others continued to smoke. A two-way table of the numbers in each group are given below with the "marginal counts" presented in the final row and column by taking row and column sums, as we did with the previous table.

	Abstain	Smoke	
Medicated	21	36	57
Unmedicated	11	44	55
	32	80	112

In modeling this situation probabilistically we use two random variables, X and Y. X is the treatment variable which has the possible values of either *Med* or *Unmed*, as the patient gets a medicated or unmedicated patch, while the Y is the outcome variable wich has the possible outcomes of *Smokes* or *Abstains.* That is:

$$\begin{aligned} X &\in \{\text{Med}, \text{Unmed}\} \\ Y &\in \{\text{Smokes}, \text{Abstains}\} \end{aligned}$$

Since this is a small sample we wouldn't know the true probabilities for the outcomes of X and Y, but we could estimate them as sample proportions as we have done in the past. Dividing the counts in the table above by the total number in the sample, 112, we get the following table of *estimated* joint probabilities, $\hat{P}(X = x, Y = y)$,

	Abstain	Smoke	
Med	.1875	.3214	.5089
Unmed	.0982	.3928	.4911
	.2857	.7143	1

For instance, from this table we can see

$$\hat{P}(X = \text{Medicated}, Y = \text{Abstain}) = .1875$$

or that

$$\begin{aligned} \hat{P}(X = \text{Med}) &= .1875 + .3214 = .5089 \\ \hat{P}(Y = \text{Abstains}) &= .1875 + .0982 = .2857 \end{aligned}$$

Perhaps the most interesting thing here is the degree to which that patch helps in quitting smoking, which can be expressed with conditional probability. For instance

$$\hat{P}(X = \text{Abstains}|Y = \text{Med}) = \frac{.1875}{.5089} = .3684$$

while

$$\hat{P}(X = \text{Abstains}|Y = \text{Unmed}) = \frac{.0982}{.4911} = .2000$$

This suggests that the patch actually helps since the abstaining rate is higher among this population, though one should wonder if the sample sizes are large enough to reach this conclusion, as we have considered before.

Example: Gender Bias

This example treats a couple of situations that consider gender bias as an explanation for our data. The existence of many kinds of biases or prejudices in people seems clearly evident in our recent and less recent history, with gender bias being only one of these. However, we take a narrower view here, looking only at the specific data sets we treat, asking what can be concluded from these alone. This kind of objectivity is essential to making good use of data, and is central to what we learn in this class. It is worth noting here that one possible outcome the analysis must consider is the *failure* to make a conclusion. This outcome would mean that the data fail to argue compellingly for gender bias, which is not the same thing as saying that the data argue for no gender bias. Statistics leaves open the possibility that we still don't know after we consider the data, which seems like a good idea.

Consider the ficititious data set presented in the table below concerning the fates of 100 job applicants to a particular technology company. Two attributes or features are measured on each applicant, their gender, as Male or Female, and the decision, Offer or Reject.

	Offer	Reject	
Male	17	51	68
Female	6	26	32
	23	77	100

We will use conditional probability notation to denote the likelihood of the two genders getting an offer from the company. That is, let

$$\begin{aligned} P(\text{Offer}|\text{Male}) &= \text{the } \textit{true} \text{ probability of a Male getting an offer} \\ P(\text{Offer}|\text{Female}) &= \text{the } \textit{true} \text{ probability of a Female getting an offer} \end{aligned}$$

where by *true* probability we imagine that there could be an infinite number of applicants xwith some proporition of them — the *true* probability — given offers. Of course, these probabilities are unknown to us, but we assume they are constant over the time of this experiment. We proceed by trying to estimate these probabilities using our data. Computing sample proportions, as usual, we get

$$\begin{aligned} \hat{P}(\text{Offer}|\text{Male}) &= \frac{17}{68} = .25 \\ \hat{P}(\text{Offer}|\text{Female}) &= \frac{6}{32} = .18 \end{aligned}$$

We see that

$$\hat{P}(\text{Offer}|\text{Male}) > \hat{P}(\text{Offer}|\text{Female})$$

Males were accepted at a higher rate in our sample. Can we conclude gender bias? We will consider three different views of this situation that deal with the scenario presented above, as well as two different variations on it, leading to different conclusions.

Case 1: Small Sample We estimated the unknown *true* probability, $P(\text{Offer}|\text{Male})$, by our sample-based estimate, $\hat{P}(\text{Offer}|\text{Male}) = .25$. Similarly, we estimated the unknown *true* probability, $P(\text{Offer}|\text{Female})$, by $\hat{P}(\text{Offer}|\text{Female}) = .18$. But what do we really know about the true probabilities? The ideas we developed before about confidence intervals apply equally well to the case at hand.

Writing $\hat{p}_M$ for $\hat{P}(\text{Offer}|\text{Male})$ and n_M for the number of males, we know that

$$\begin{aligned} \hat{p}_M \pm 1.96\sqrt{\frac{\hat{p}_M(1-\hat{p}_M)}{n_M}} &= .25 \pm 1.96\sqrt{\frac{(.25)(.75)}{68}} \\ &= .25 \pm .10 \end{aligned}$$

is a 95% condfidence interval for the true probability, $P(\text{Offer}|\text{Male})$. Similarly, writing $\hat{p}_F$ for $\hat{P}(\text{Offer}|\text{Female})$ and n_F for the number of females, we get

$$\begin{aligned} \hat{p}_F \pm 1.96\sqrt{\frac{\hat{p}_F(1-\hat{p}_F)}{n_F}} &= .18 \pm 1.96\sqrt{\frac{(.18)(.82)}{32}} \\ &= .18 \pm .13 \end{aligned}$$

is a 95% condfidence interval for the true probability, $P(\text{Offer}|\text{Female})$. These intervals describe the reasonable range of values for the true probabilities. While the center of the female interval, .18, is lower than the center of the male interval, .25, we can see that these intervals *overlap*. In particular, every probability that lies between the lower endpoint of the male interval, .15, and the upper endpoint of the female interval, .31, is in *both* confidence intervals, so is a plausible offer rate for *both* male and female applicants. Thus, for instance, it is plausible that

$$\begin{aligned} P(\text{Offer}|\text{Male}) &= .15 \\ P(\text{Offer}|\text{Female}) &= .31 \end{aligned}$$

which is a situation where women are given job offers at a *higher* rate than men. Since the intervals overlap, there are, of course, many other pairs of reasonable values (values inside their respective confidence intervals) where the women are given job offers at a higher rate than men. We cannot conclude that $P(\text{Offer}|\text{Male}) > P(\text{Offer}|\text{Female})$ from this sample because the confidence intervals overlap. Since we can't even conclude that men are given offers at a greater rate than women, we can't possibly conclude any kind of gender bias here.

Case 2: Observational Samples In the situation we just studied, we couldn't make any conclusion about gender bias because our sample was too small. We did not conclude that there was no gender bias, we just failed to make any conclusion at all. Suppose instead that we had a much larger sample. Say, instead of 68 men and 32 women we had 6,800 men and 3,200 women, but with the same values of $\hat{P}(\text{Offer}|\text{Male}) = .25$ and $\hat{P}(\text{Offer}|\text{Female}) = .18$. In this case, having multiplied the sample sizes by 100 we have decreased the confidence intervals widths by a factor of $\sqrt{100} = 10$ (do this computation yourself), thus giving $.25 \pm .010$ for the male interval and $.18 \pm .013$ for the female interval. These intervals no longer overlap. Thus any plausible choice for $P(\text{Offer}|\text{Male})$ is greater than any plausible choice for $P(\text{Offer}|\text{Female})$. We *can* conclude that $P(\text{Offer}|\text{Male}) > P(\text{Offer}|\text{Female})$.

The data say that men are given offers at a higher rate than women, but is this the same as concluding gender bias? Perhaps there is some other trait — a trait we didn't measure — that occurs in higher proportion in the male applicant popluation. It could be that the employers respond to this trait, rather than to the gender itself. For instance, it may be that, for whatever reason, the male applicants tend to be older than the female applicants. This could be due to many possible explanations such as a greater time, on average, for the men to finish the appropriate schooling, or later time, on average, for the men to seek financial independence, thus applying for jobs. It could be that the employer's preference for older applicants ends up favoring the male applants, not because they are *male*, but because they are *older*. One can imagine many other traits that would occur in different proportions between men and women, that, when sought by the employer, would create a preference for men over women. Thus we cannot conclude gender bias in this case. That is, though we *can* conclude that male candidates are preferred over female candidates, we *cannot* conclude that they are prefered *because* they are male applicants.

This is a common situation that arises in what is called an *observational study*. In an observational study we do not put our subjects into the the various categories we measure — we didn't assign a gender to the applicants. We just observed the genders of the people who applied. Instead of an observational study one could do a *controlled study* in which we *assign* the "explanatory" variables ourselves. For instance, in the case we discussed earlier with smokers and the medicated patch, we could assign the choice of medicated vs. unmedicated patch to the subjects ourselves. Usually this assignment is done at random so it cannot be connected with any of the other variables we measure. It would seem to be impossible to do a controlled study with our job applicants since it would involve assigning gender through the study. Clearly we can't do that. Or can we . . . ?.

Case 3: Controlled Study In the previous case we said we couldn't assign the genders ourselves, and this is obviously true, though we can do something that is very close to this. Suppose we have a *single* job applicant with a resume. The applicant sends the resume to a number of different employers with only one difference between the resumes: for half of the employers the name of the resume is chosen to be a traditionally female name, say "Jane," while for the other half the name is chosen to be traditionally male, say "John." We make this choice at random so it isn't connected with any attribute of the different employers.

Now we compute from the two samples and write $\hat{P}(\text{Offer}|\text{Male})$ and $\hat{P}(\text{Offer}|\text{Female})$ for the observed proportion of offers from employers that were sent the "Male" and "Female" applicants. This is identical notation to what we used before, though the sitution is somewhat different since we are conditioning on the employer's *belief* of gender, not the *actual* gender. Suppose we find that

$$\hat{P}(\text{Offer}|\text{Male}) > \hat{P}(\text{Offer}|\text{Female})$$

and that the sample sizes were large enough that the result is statistically significant — the confidence intervals don't overlap. Can we now conclude gender bias?

Suppose we divided our employers into two, but didn't play our trick with the name on the application, instead giving the two employer groups the *identical* resume. Since we divided the employers at random we can consider the two groups to be statistically identical. This does not mean that we have two indentical copies, since, of course, we have different employers in the two different groups. Rather it means that they can be viewed as two samples from the same population. Any differences between two samples can only be attributed to the random variation of sampling — not any systematic difference. When we include the two different names in the resume, we are sampling from two populations whose *only* difference is the name given to the resume. Since the two popluations are identical in every other regard, any statistically signficant differences between the two popluations can only be attributed to the resume's name. Since we have found a statistically significant difference between the two groups we conclude that the difference in offer rate was *caused* by the difference in names — this is the only possible explanation. Thus we *can* conclude gender bias.

1.6 Independence

Informally, events A, B are *independent* if knowledge of one tells nothing about the other. For example suppose we roll a pair of dice and consider the events

$$\begin{aligned} A &= \text{1st die is 6} \\ B &= \text{2nd die is 6} \end{aligned}$$

What happens with the first die tells nothing about what happens with the second die, so these events should be independent.

Our notion can be made more precise, as follows. If A, B are independent events then *conditioning* on one event shouldn't change the probability of the other. That is,

$$\begin{aligned} P(A|B) &= P(A) \\ P(B|A) &= P(B) \end{aligned}$$

In practice, we can show that either of these equations demonstrates independence. That is, if the first condition holds then so does the 2nd one, and vice-versa. So either of these can be our *definition* of independence. In the

previous section we discussed conditional probability for random variables, though this applies just as well to events. That is,

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Let's consider some examples.

Example (Independence of Dice) Consider a pair of dice and let A be the event that the first die is a 6, with B be the event that the second die is a 6. Intuitively, A, B should be independent either the die "doesn't know" what the other die is doing. But let's do this by calculation. Using our definition of conditional probability

$$\begin{aligned} P(A) &= \frac{1}{6} \\ P(A|B) &= \frac{P(A \cap B)}{P(B)} = \frac{1/36}{1/6} = \frac{1}{6} \end{aligned}$$

Since $P(A) = P(A|B)$ the events are independent.

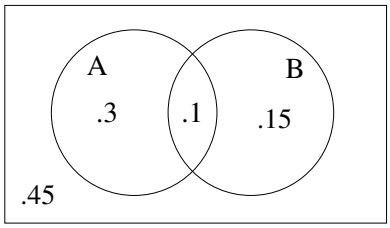
Example (Cash behind one of 3 doors) Suppose that some cash is hidden behind one of 3 different doors. Consider the events:

$$\begin{aligned} A &= \text{\$ not behind door \# 1} \\ B &= \text{\$ not behind door \# 2} \end{aligned}$$

Are A and B indepedent? Intuitively these should not be independent because the money has to be behind one door, so if we know the money is not behind door #1 (A) then it is more likely to be behind door #2 (B^c). From a computational perspective $P(A) = 2/3$, but $(A|B) = 1/2$. Since $P(A) \neq P(A|B)$ these events are not independent.

Example (Using Venn Diagram) Consider the example below where the A and B are the two circled regions while the numbers give the probabilities for the 4 subregions:

$$\begin{aligned} P(A \cap B) &= .1 \\ P(A \cap B^c) &= .3 \\ P(A^c \cap B) &= .15 \\ P(A^c \cap B^c) &= .45 \end{aligned}$$



Are A and B independent here? From the numbers we see that

$$\begin{aligned} P(A) &= P(A \cap B) + P(A \cap B^c) = .1 + .3 = .4 \\ P(B) &= P(B \cap A) + P(B \cap A^c) = .1 + .15 = .25 \\ P(A|B) &= \frac{P(A \cap B)}{P(B)} = \frac{.1}{.25} = .4 \end{aligned}$$

Since $P(A) = P(A|B)$ A and B are independent.

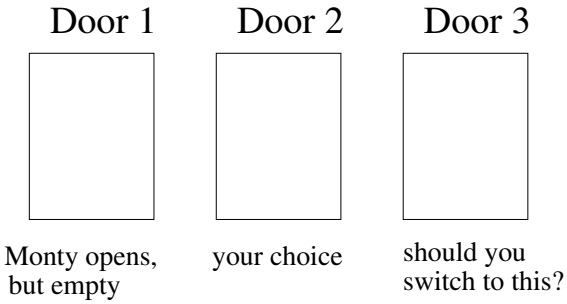
Example (Coin flips with random probability of heads) Let p be the probability of flipping heads. Choose p at random between 0 and 1, (e.g. runif(1)), and flip twice with this p. Let

$$\begin{aligned} A &= \text{1st flip H} \\ B &= \text{2nd flip H} \end{aligned}$$

Are A and B independent? The example two_flips_random_p.r simulates the situation and indicates that events are not independent. Intuitively, A gives information about p which, in turn, gives information at B. That is, if A occurs we think that higher values of p are more likely, so that B is more likely to occur. Conversely, if A does not occur than lower values of p are more likely, so B is less likely to occur. This intiuition suggests that A, B not independent, which is borne out by our simulation.

The Monty Hall Problem

In a famous game show from years past a prize is placed behind one of three doors. The contestant chooses a door, but *does not* open the door. The host of the show, Monty Hall, who knows which door contains the prize, chooses a door known to *not* contain the prize. Monty then offers you the choice of switching your door to the remaining door. For instance, consider the picture below:



You begin by choosing door 2, after which Monty reveals door 1 to be empty. Monty offers you the choice of switching to door 3. Should you do it?

We will simulate this experiment to estimate the probabilities of various outcomes in the R example monty_hall.r, but before we do this think about the problem and decide if you think it is worth switching to the remaining door. It is often instructive to do this, even if you don't make the right conclusion.

This is a rather strange example that defies many people's intuition. In fact, the example was the subject of a great deal of debate in a math trivia newspaper column by Marilyn vos Savant, where a number of educated professors vigorously defended the wrong answer. In fact, it is a little subtle and depends on the exact way the problem is posed, so I was careful in the way I did this. Let's now consider the two possible strategies of **Keep** and **Switch** to see what happens in each.

The **Keep** strategy, where you remain with your originally chosen door, is the easiest to analyze. Here you choose a door at random and win only if that choice was correct, so $P(\textbf{Keep}\text{ wins}) = 1/3$. For the **Switch** strategy, on the other hand, your intial choice will either be right or wrong. If the initial choice is correct and you switch your door you will lose. However, if your intial choice is wrong, Monty will reveal the remaining losing door, so switching will bring you to the correct door. Thus the switch strategy wins when your initial choice is wrong and loses when your initial choice is right. Thus $P(\textbf{Switch}\text{ wins}) = P(\text{Initial choice wrong}) = 2/3$.

Conditional Probability Revisited Suppose A, B events with probabilities $P(A)$, $P(B)$ and joint probability $P(A \cap B) = P(A \text{ and } B)$. Recall the reasoning from the following table:

	A	A^c	
B	.1	.2	.3
B^c	.3	.4	.7
	.4	.6	

If we want to get the conditional probability, $P(B|A)$ we simply need to restrict our attention to the "world" where A occurs, and normalize the probabalities in that world to sum to 1. For example, from the table $P(B|A) = .1/.4 = .25$. This led to the definition that lead to the definition

$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$

If we multiply the above equation on both sides by $P(A)$ to get

$$P(A \cap B) = P(A)P(B|A)$$

which is a useful and intuitive formula. For example, suppose we have a drawer with 2 red and 4 black socks. What is the probability of drawing a pair of black socks? Let

$$\begin{aligned} A &= \text{1st sock black} \\ B &= \text{2nd sock black} \end{aligned}$$

Then

$$P(\text{both socks black}) = P(A \cap B) = P(A)P(B|A) = \frac{4}{6}\frac{3}{5} = \frac{2}{5}$$

Independence Revisited We defined events A and B to be independent if $P(A|B) = P(A)$ or, the other way around, $P(B|A) = P(B)$. Now using the discussion in the section above, if A and B are independent we get

$$P(A \cap B) = P(A)P(B|A) = P(A)P(B)$$

which is the *usual* definition you will see in most books — for independent events the probability of *both* occuring is the product of the probabilties of *each* occuring.

For instance, suppose we have a table of probabilities for independent events and we only know the *marginal* probabilities, as below

	A	A^c	
B			.4
B^c			.6
	.2	.8	

From this information, *assuming independence* we can fill in the entire table using the product rule:

	A	A^c	
B	.08	.32	.4
B^c	.12	.48	.6
	.2	.8	

Bayes' Rule We introduce Bayes' rule informally with an example. Suppose a certain town has two factories, 1 and 2, with everyone in the town working at one of these factories. Suppose factory 1 is bigger with 80% of the town's residents working there. At a town breakfast both pancakes and waffles are served. We know that 70% of people from factory 1 prefer waffles to pancakes, while 70% of people from factory 2 prefer pancakes to waffles. A randomly selected person at the breakfast is found to eat pancakes. What is the probability this person works for factory 1?

One way to do this would be to begin by filling in the table for joint probability. We could begin this by including the marginal probabilities for the 2 factories, as done below.

	Factory 1	Factory 2	
Pancakes			
Waffles			
	.8	.2	

Next we could fill in the middle of the table according to our rule for conditional probability: $P(A \cap B) = P(A)P(B|A)$. For instance,

$$\begin{aligned} P(\text{Factory 1} \cap \text{Pancakes}) &= P(\text{Factory 1})P(\text{Pancakes}|\text{Factory 1}) \\ &= (.8)(.3) \\ &= .24 \end{aligned}$$

since 30% of factory 1 workers perfer pancakes. Continuing in this manner gives

	Factory 1	Factory 2	
Pancakes	.24	.14	
Waffles	.56	.06	
	.8	.2	

Finally we can get the marginal probabilityes on Pancakes and Waffles by taking row sums:

	Factory 1	Factory 2	
Pancakes	.24	.14	.38
Waffles	.56	.06	.62
	.8	.2	

Now, reasoning from the table we get

$$P(\text{Factory 1}|\text{Pancakes}) = \frac{.24}{.38} = .63$$

More formally, suppose A, B are two events and we know the probability A occurs, $P(A)$, as well as the probability of B when A occurs, $P(B|A)$ and when A does not, occur, $P(B|A^c)$. Bayes' rule allows us to compute $P(A|B)$ by using our definition of conditional probability and writing out some things we already know:

$$\begin{aligned} P(A|B) &= \frac{P(A \cap B)}{P(B)} \\ &= \frac{P(A \cap B)}{P(A \cap B) + P(A^c \cap B)} \\ &\overset{\text{B.R}}{=} \frac{P(A)P(B|A)}{P(A)P(B|A) + P(A^c)P(B|A^c)} \end{aligned}$$

The last line is known as Bayes' rule. For instance, revisiting the problem that we began with

$$\begin{aligned} P(\text{Factory 1}|\text{Pancakes}) &= \frac{P(\text{Factory 1})P(\text{Pancakes}|\text{Factory 1})}{P(\text{Factory 1})P(\text{Pancakes}|\text{Factory 1}) + P(\text{Factory 2})P(\text{Pancakes}|\text{Factory 2})} \\ &= \frac{(.8)(.3)}{(.8)(.3) + (.2)(.7)} \\ &= .63 \end{aligned}$$

Law of the Rare Disease An interesting application of Bayes' rule is to a case known as the *Law of the Rare Disease*, though it is really a generic example that occurs in many different situations. Suppose we have a rare disease that is present in .1% of the population. We have a test for the disease that is fairly accurate. In particular, suppose that when a person has the disease the test gives a positive result 99% of time time. Similarly, when the person does *not* have the disease, the test gives a negative result 99% of the time. The test seems pretty good.

A random person is chosen who tests positively for the disease. Is it likely the individual has the disease? You should consider this and commit yourself to an answer before reading on. All relevant information is in the paragraph above; there is no trick here.

For notation, let

$$\begin{aligned} D &= \text{disease present} \\ + &= \text{person tests positive} \\ - &= \text{person tests negative} \end{aligned}$$

From the description of the problem we have

$$\begin{aligned} P(D) &= .001 \\ P(D^c) &= .999 \end{aligned}$$

and

$$P(+|D) = P(-|D^c) = .99$$

By Bayes' rule

$$\begin{aligned} P(D|+) &= \frac{P(D)P(+|D)}{P(D)P(+|D) + P(D^c)P(+|D^c)} \\ &= \frac{(.001)(.99)}{(.001)(.99) + (.999)(.01)} \\ &\approx \frac{1}{11} \quad \text{(approximating both .99 and .999 as 1)} \end{aligned}$$

It turns out to be pretty unlikely the person has the diesease even when we get a positive test. If this seems counter-intuitive, look at the denominator in the fractions above. The denominator shows two ways to get a positive result

1. The person has the disease and tests positive. This happens with probability (.001)(.99)

2. The person does not have the disease and tests positive. This happens with probability (.999)(.01)

The 2nd explanation is about 10 times more likely than the first explanation. When the explanations are viewed as conditional probabilities, they must sum to 1. $\frac{1}{11}$ and $\frac{10}{11}$ are two numbers that sum to 1 with the second number being 10 times bigger than the first, so these must be the *conditional* probabilities of having/not having the disease given a positive test result.

This is illustrated computationally in law_of_rare_disease.r.

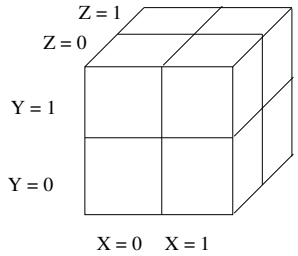
1.7 Tables

First we begin by reviewing something we have already seen. Suppose we observe two random variables, X, Y, reporting two features or attributes we measure from some population. If X can occur in n ways and Y can occur in m ways, the result could be summarized by an $n \times m$ table of counts:

	$Y=1$	$Y=2$	$\ldots$	$Y=m$
$X=1$				
$X=2$				
$\vdots$				
$X=n$				

That is, in the x, y element of the table (xth row, yth column) we would have the number of times the event $(X = x, Y = y)$. An example would be the 2×2 on smoking and treatment we saw earlier.

Similarly, if X, Y, Z are *three* random variables, we could store our results in a 3-way table. As a special case, suppose that X, Y, Z are binary variables. In this case we could think of the table as a $2 \times 2 \times 2$ cube with a total of 8 cells, as pictured below:



In this table we would have, for instance, $\#(X = 1, Y = 0, Z = 1)$ in the (1,0,1) element of this table. Relating this to computation, it would be natural to think of the table as a $2 \times 2 \times 2$ array, c, having 8 possible cells. In this case it would take 3 binary indices to specify a cell in the table, so, for instance, $c[1, 0, 1]$ is the number of times the event $(X = 1, Y = 0, Z = 1)$ occured in the sample.

There are two operations on tables we introduce now, *marginalization* and *conditioning*. These should seem familiar, since they are analogous to the marginal and conditional probability distributions we have already seen.

Marginalization Suppose we ask a collection of people about their ice cream preference, summarized in the following table.

		Flavor		
		Chocolate	Vanilla	Neither
Age	Children	40	22	15
	Teens	12	16	45
	Adults	55	54	10

Here we have a two variables, Flavor and Age, with

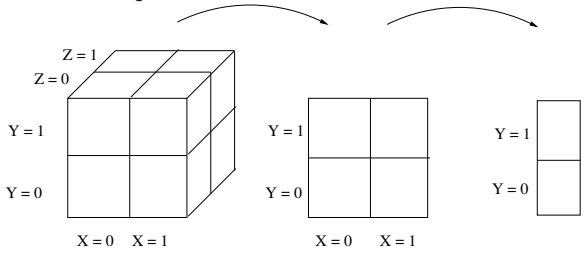
$$\begin{aligned} \text{Flavor} &\in \{\text{Chocolate}, \text{Vanilla}, \text{Neither}\} \\ \text{Age} &\in \{\text{Children}, \text{Teens}, \text{Adults}\} \end{aligned}$$

If we were to only ask about flavor preference we would get the one-dimensional table given below by taking column sums.

Flavor		
Chocolate	Vanilla	Neither
107	92	70

This is the *marginal* table on Flavor. Sometimes people say we *marginalize* to the Flavor variable, or we *marginalize out* the Age variable. This is analogous to how we compute row sums and column sums with probability tables to get marginal probability distributions.

As mentioned before, if we measured 3 binary variables X, Y, Z we would get a 3-dimensional $2 \times 2 \times 2$ table. We could marginalize down to the X, Y variables by summing out the Z variable, resulting in a 2×2 table. We could *further* marginalize down to the Y variable by summing out the X variable of the 2×2 table, resulting in a 2×1 table. These operations are pictured below.

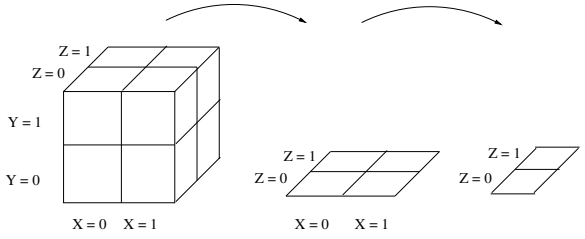


Conditioning We can also produce tables from other tables by *fixing* or *conditioning* on a specific value of a variable. For instance, if we were to condition on Flavor = Vanilla in the 3×3 ice cream table we get a the table on Age below

	Children	22
Age	Teens	16
	Adults	54

Here we have simply restricted our attention to the world where Flavor = Vanilla, resulting in a 3×1 table.

For the case of the 3 binary variables, X, Y, Z we could first condition on $Y = 0$ to get the 2×2 table on X, Z. This is just the "bottom slice" of the original table. We could further restrict by conditioning on $X = 1$ to get the 1-variable table on Z. One could either view this as "bottom-right" quadrant of the original table, or just as the "right" half of the 2nd table. These operations are shown below:



We could potentially have more than three variables in our tables, and could conceivably mix conditioning and marginalization in many different ways. For instance, a 4-variable table could be reduced to a 1-dimensional table on the first variable by fixing the value of the 4th variable and marginalizing out the 2nd and 3rd variables. **But for there to be any point in doing such operations we will need to be able interpret the meaning of the resulting tables.** We will take this up in what follows.

1.7.1 Tables in R

We begin by looking at the example table_margin_condition.r that demonstrates some basics of handling tables in R. Since this example introduces a number of new aspects of the R language, we will discuss the example in more detail than some of the other R examples. The table_margin_condition.r program treats the famous UCB (= University of California at Berkeley) 1973 graduate admissions data. You can import the dataset into R with the line

```
data("UCBAdmissions")
```

As with a number of other useful datasets, R just "knows about" this dataset automatically, so you don't need to download it or tell R where it is on your computer. Since the name "UCBAdmissions" is long, we will use the R command

```
ucb = UCBAdmissions
```

so we don't need to keep typing the longer name over and over in what is to come. The command

```
dimnames(ucb)
```

prints out a description of the dataset, as below

```
$Admit
[1] "Admitted" "Rejected"

$Gender
[1] "Male"   "Female"

$Dept
[1] "A" "B" "C" "D" "E" "F"
```

The meaning is that there are three variables in the dataset: Admit, Gender, and Dept, with

$$\begin{aligned} \text{Admit} &\in \{\text{Admitted}, \text{Rejected}\} \\ \text{Gender} &\in \{\text{Male}, \text{Female}\} \\ \text{Dept} &\in \{A, B, C, D, E, F\} \end{aligned}$$

The data come from one particular year of UCB graduate school admissions in which a collection of students applied to one of six different departments, known here simply as A through F. The dataset measures three attributes on each student, their admission status, gender, and chosen department. As a word of caution, the dimnames command works just as I used it if you type it at the command line, but, as with other R commands that print text output, you need to write

```
print(dimnames(ucb))
```

in your R script if you are going to run a collection of lines from a file.

The ucb object is a *table*, much like the ones we have already discussed in class, except that there are 3 variables in the table, so it is 3-dimensional. In R a table is very much like an array or matrix, still allowing the operations we have already learned. For instance, you could refer to the individual counts of table as

```
ucb[1,1,1]
```

to get the number of Male students who were Admitted to department A. (Admitted, Male, and A are the first value for each variable in the printout of dimnames, hence the three 1 indicies.) If you think it is cryptic to refer to the possible values of the variables by numbers you can also write

```
ucb["Admitted","Male","A"]
```

to get the same count. You could also get the 2×2 table of counts on the applicants to a specific department — say department A by

```
ucb[,,"A"]
```

leaving the first two indices blank. Or the 1-d table of counts on students who were Admitted to department A by

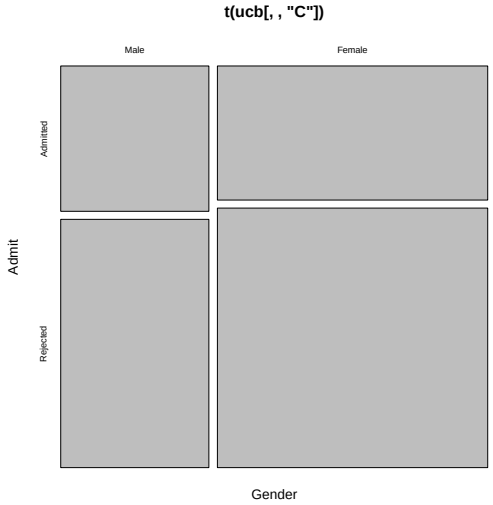
```
ucb["Admitted",,"A"]
```

This is the concept of *conditioning* already discussed.

Tables of counts are rarely easy to absorb when viewed as raw numbers, so we will usually visualize 2-dimensional tables with the mosaicplot command, for instance

```
mosaicplot(ucb[,,"C"])
```

resulting in a graphical depiction of the 2×2 table on Gender and Admit status for applicants to department C, shown in output as depicted below:



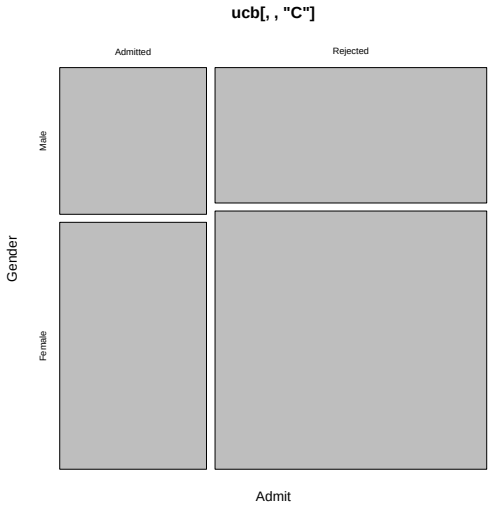
The mosaicplot shows a plot of a 2-dimensional table (a matrix) where the columns correspond to the first variable and the rows correspond to the 2nd variable. This is the reverse of the usual matrix indexing, but is the convention used by R. In a mosaicplot

- The area in each rectangle is proportional to the associated count.
- The widths of the columns in the plot are proportional to the total count for each column.

Thus we can easily see from the plot that more Female students applied to department C (the Female column is wider), and that the biggest of the four counts for department C is Rejected Female students (the area is the largest). If we wanted to reverse the plot so that we have Admit status for the columns and Gender in the rows, we only need to "transpose" (that is, flip) the table. This is done with the R command t():

```
mosaicplot(t(ucb[,,"C"]))
```

which produces that output below.



Note that the rows and columns of the plot are reversed.

R also allows us to *marginalize* our tables. For instance, if we wanted to get the 1-dimensional table on the Dept variable (without regard for Admit status or Gender) we would use

```
apply(ucb,"Dept",sum)
```

This is admittedly a little cryptic when you see it for the first time, but remember we get a marginal table by summing out the variables we don't want, so we "apply" the "sum" operation to the table. If we wanted to get the 2-dimensional table on Department and Gender we would use

```
apply(ucb,c("Dept","Gender"),sum)
```

You should treat the above as a tutorial introduction to tables in R and try these commands out for yourself.

Practice

1. Make a mosaic plot on Admission and Department without considering the Gender variable.

2. Make a mosaic plot on Admission and Department for the Female candidates.

3. Print out the one-dimensional table on Department for the Female Candidates.

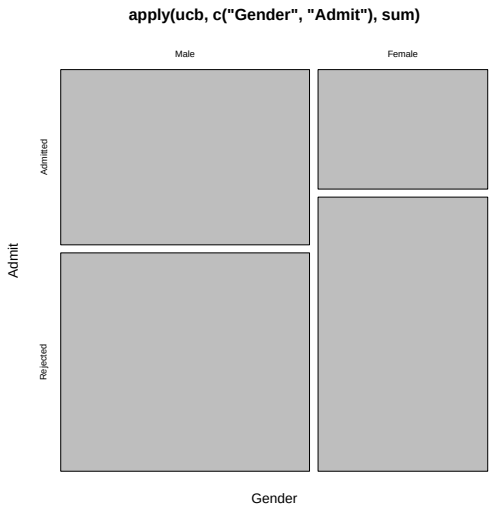
1.7.2 Simpson's Paradox

The UCB admission data is famous as an example of Simpson's paradox. This is an idea we have already begun to discuss, in which we seem to see a connection or cause-and-effect between two variables when examined in isolation, though this connection may not continue to exist when examined in the context of another variable.

To be more explicit, consider the code fragment below that acts on the UCB admission data, taken from the simpsons_paradox.r program.

```
data("UCBAdmissions");                          # import the data
ucb = UCBAdmissions;                            # abbreviate "UCBAdmissions"
mosaicplot(apply(ucb,c("Gender","Admit"),sum)); # mosaic plot clearer ...
```

The code produces a plot of Gender vs. Admit status as shown below. In doing this we have summed all of the departments together. This is what would happen if we didn't even ask about department and just collected our data on Gender and Admit. Said another way, we are visualizing the *marginal* distribution on Gender and Admit.



We now consider this figure in the context of the familiar formula

$$\hat{P}(A \cap B) = \hat{P}(A)\hat{P}(B|A)$$

where, for example, A is the event that the student is Male and B is the event that the student is Admitted. Recall that the widths of the rectangles are proportional to the column counts, which are, in turn, proprotional to the estimated Gender probabilities. So, for instance, from the figure we can see that $\hat{P}(A) \approx .6$ since the Male rectangle accounts for about 60% of the total width. Also recall that the areas of the rectangles are proportional to cell counts, which are, in turn, proportional to estimated joint probabilities. Thus, for instance, we see that $\hat{P}(A \cap B) \approx .25$ since the (Male, Admitted) cell accounts for about a quarter of the total area.

But what is the meaning of a rectangle height? We know that

$$\text{Area} = \text{Width} \times \text{Height}$$

and have already seen rectangle Area as $\hat{P}(A \cap B)$, and rectangle Width as $\hat{P}(A)$. Thus, matching terms in the above two formulas it must be that Height is proportional to $\hat{P}(B|A)$. The heights represent *conditional* probabilities, conditioned on the column variable. This is the key to interpreting the mosaicplot.

Now to Simpson's paradox, reasoning from this figure we see that

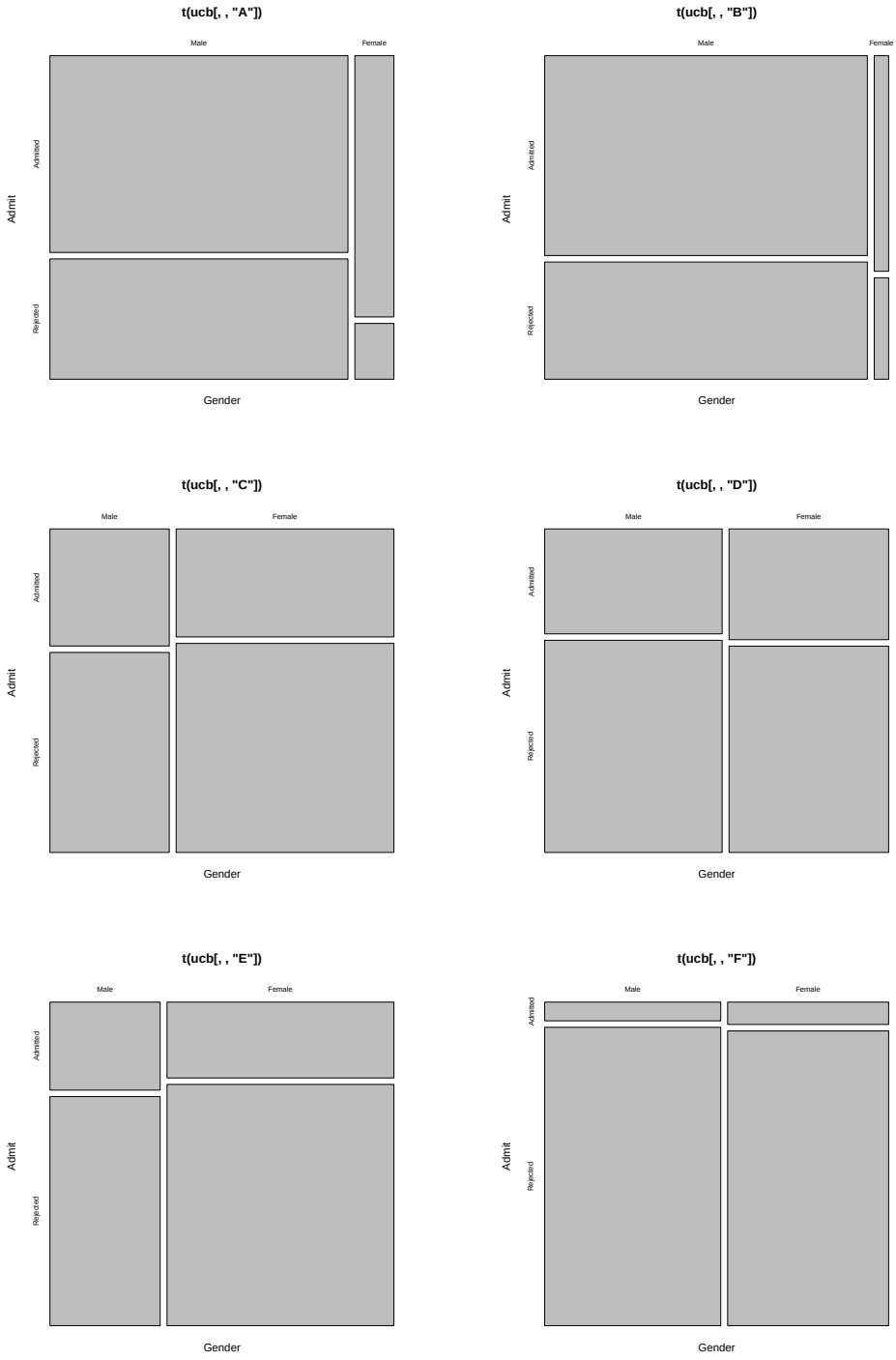
$$\hat{P}(\text{Admitted}|\text{Male}) > \hat{P}(\text{Admitted}|\text{Female})$$

since the height of the (Male, Admitted) rectangle is taller than the height of the (Female, Admitted) rectangle. Also observe that the sample size is large enough that this difference *statistically significant* — that is, the result is not just chance variation due to small sample size. After all, we have nearly 2000 Male students and more Female students, so, by the $\frac{1}{\sqrt{n}}$ rule our errors are on the order of .02, while our estimated conditional probabilities differ by several times this error margin.

In our first encounter with this situation we tried to *imagine* other variables whose existence might explain the difference in acceptance rate between Males and Females, but in this case we don't need to do this. The data set *gives* us the important variable — the Department. Suppose we now look at the "differential" acceptance rates between Males and Females for each Department, as done in the R code below

```
mosaicplot(t(ucb[,,"A"]));  # department A seems to *favor* Females
mosaicplot(t(ucb[,,"B"]));  # B seems to have a slight bias for Females (could be neutral)
mosaicplot(t(ucb[,,"C"]));  # others seem neutral
mosaicplot(t(ucb[,,"D"]));
mosaicplot(t(ucb[,,"E"]));
mosaicplot(t(ucb[,,"F"]));
```

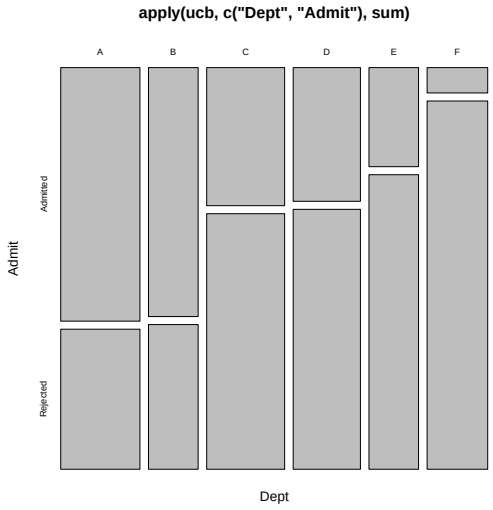
and in the following plots



It seems that each department either *favors* Females or is neutral. But when we look at the data in aggregate, as in our initial plot, there appears to be a bias *against* Females. **How can this be?**

Part of the answer lies in the difference between the Departments. The R code and the plot below show the different acceptance rates between the departments, summing out over gender.

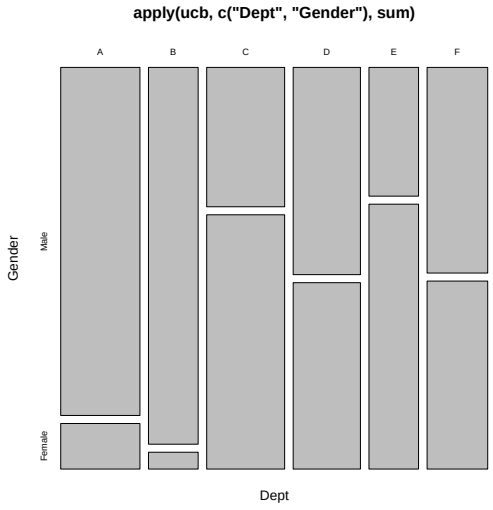
```
mosaicplot(apply(ucb,c("Dept","Admit"),sum));  # the deptments' acceptance rates differ
```



Here the order of the departments becomes clear. They are listed from the easiest (A) to the hardest (F) in terms of their acceptance rates. You may be surprised to learn that the departments with the highest acceptance rates were Engineering and Chemistry while the hardest department to get into was English.

As it turns out, shown below, the Male applicants were targeting to the easier departments.

```
mosaicplot(apply(ucb,c("Dept","Gender"),sum));  # men apply more to easier depts A and B
```



So the higher acceptance rate for the male applicants comes from the way that the Males tended to apply to departments with higher acceptance rates than the Female candidates did.

The moral is that one needs to be careful about inferring cause and effect when looking at observational studies such as this one. In this case we were lucky enough to find a variable, the Department, that "explains away" the seeming cause-and-effect between Gender and Admit. But if there is no such variable in our data set, it is always possible that we simply failed to identfy and measure it. It will always be difficult to prove a cause and effect relationship from observational data.

1.7.3 Titanic Survival Data

R also knows about a dataset concerning the Titanic disaster. Here we will use this dataset to consider the survival rates of different groups on the ship. The following code, with its output, shows the structure of the data

```
data(Titanic)            # R just knows about the Titanic data too
print(dimnames(Titanic)) # the variables and their categorizations
```

```
$Class
[1] "1st"  "2nd"  "3rd"  "Crew"

$Sex
[1] "Male"   "Female"

$Age
[1] "Child" "Adult"

$Survived
[1] "No"  "Yes"
```

We see that there are four different variables, Class, Sex, Age, Survived with

$$\begin{aligned} \text{Class} &\in \{1\text{st}, 2\text{nd}, 3\text{rd}, \text{Crew}\} \\ \text{Sex} &\in \{\text{Male}, \text{Female}\} \\ \text{Age} &\in \{\text{Child}, \text{Adult}\} \\ \text{Survived} &\in \{\text{No}, \text{Yes}\} \end{aligned}$$

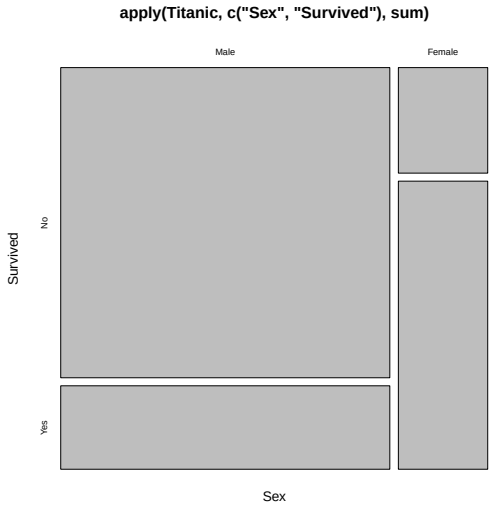
Thus we have a table with $4 \times 2 \times 2 \times 2 = 32$ possible counts. From the Titanic data we can produce a number of other tables. For instance

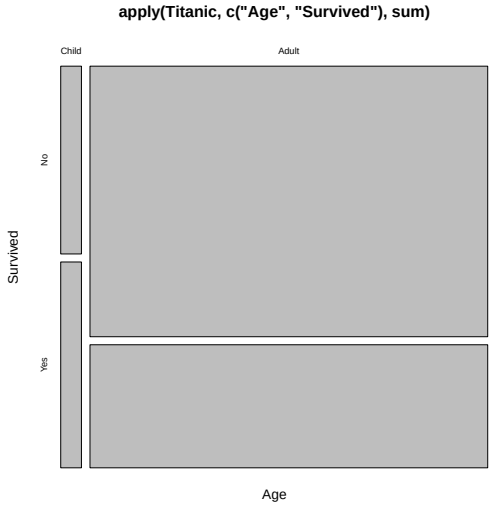
```
print(apply(Titanic, c("Sex","Survived"),sum));  # table on  Sex and Survived
print(apply(Titanic["1st",,,],c("Sex","Survived"),sum))   # table on Sex and Survived for the 1st class
```

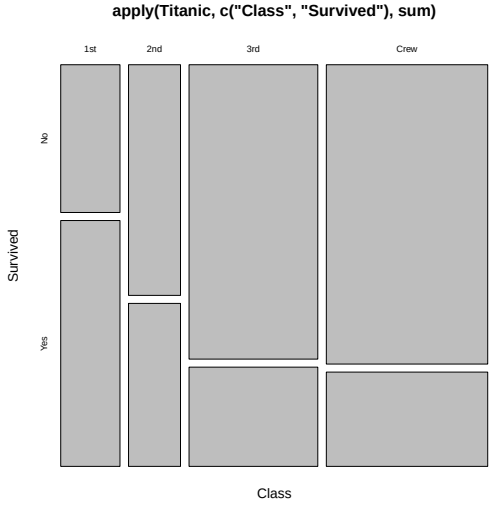
show ways to get the two-way table on Sex and Survived, as well as table on those same to variables *restricted* to the 1st class passengers. These examples are just for practice.

As discussed before, plots are much easier to intepret. The code below generates mosaicplots on three pairs of variables: Survival and Sex, Survial and Age, Survival and Class:

```
mosaicplot(apply(Titanic, c("Sex","Survived"),sum));  # table on  Sex and Survived
mosaicplot(apply(Titanic, c("Age","Survived"),sum));  # table on  Age and Survived
mosaicplot(apply(Titanic, c("Class","Survived"),sum));  # table on  Class and Survived
```


apply(Titanic, c("Sex", "Survived"), sum)
Male
Female
Survived
No
Yes
Sex

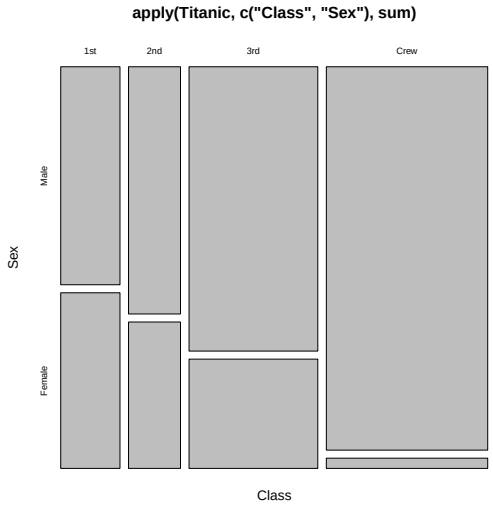

apply(Titanic, c("Age", "Survived"), sum)
Child
Adult
Survived
No
Yes
Age


apply(Titanic, c("Class", "Survived"), sum)
1st
2nd
3rd
Crew
Survived
No
Yes
Class

From these we can see that women survived in greater proportion than did men. Similarly we can see that children survived in greater proprotion than adults. So it seems that the "Women and children first" direction was followed here. But it is also suprising that the higher class passengers also survived in higher proportion than the lower class passengers, as the last mosaic plot makes clear, since we haven't heard of any official stated preference for them. In what follows we will no longer consider age, since there were too few children on board to come to any reliable conclusions.

There seems to be an interesting connection between gender and class, as shown below.

```
mosaicplot(apply(Titanic, c("Class","Sex"),sum));  # table on  Class and Sex
```



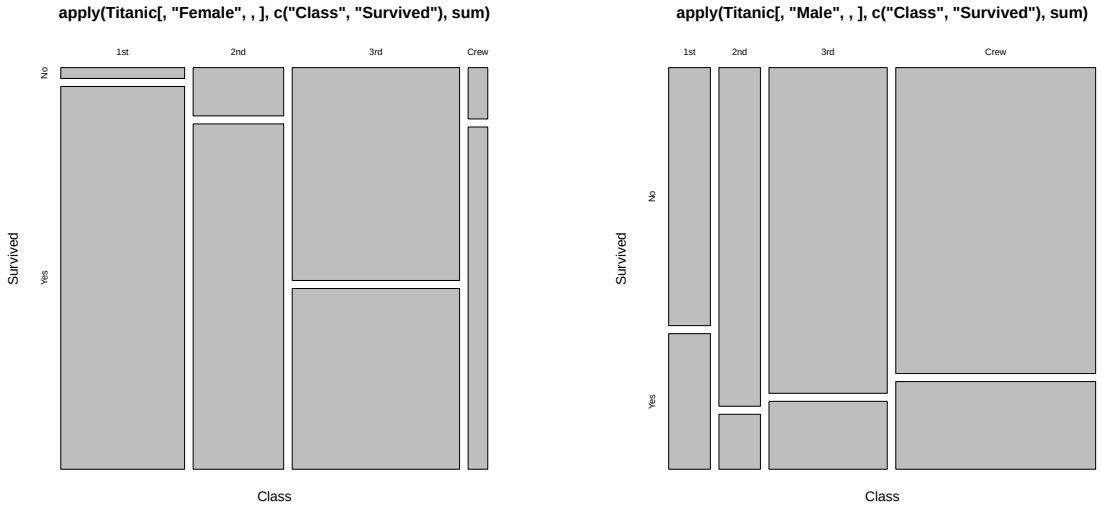
In this plot we see that the women are more represented in the higher classes. This brings up an interesting alternative.

1. Could it be that the survival preference for higher classes just comes because women were prefered and there are more women in the higher classes?

2. Or could there be an additional preference for higher classes without regard for gender?

We can examine this question by looking at the two-way table on Class and Sex, as above, *restricting* to the different genders. If, considering the genders separately, we still see a survival preference for 1st class passengers, this preference must be genuine. The code below shows the generation of these *restricted* tables, as well as the resulting plots.

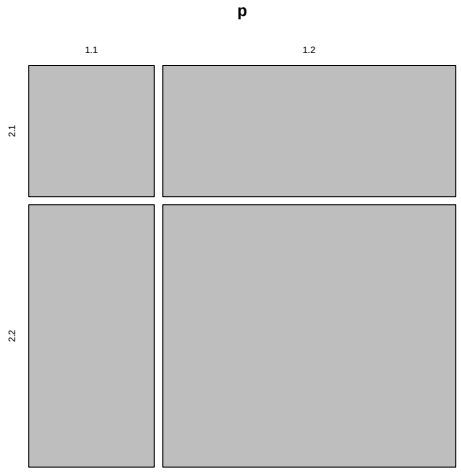
```
mosaicplot(apply(Titanic["Female",,], c("Class","Survived"),sum));  # table on  Sex and Survived
mosaicplot(apply(Titanic["Male",,], c("Class","Survived"),sum));  # table on  Sex and Survived
```



Within each gender there still is a survival preference for first class passengers, so this cannot be attributed to the way the genders are distributed among the classes. It appears that in addition to the preference for women and children, there was also a preference for first class passengers.

1.8 Independence and Conditional Independence

Consider the mosaic plot below of two variables A, B. Here the mosaic plot represents the joint probability of A, B rather than counts from a sample.



From the plot are A and B independent? Since the rectangle heights correspond to conditional probabilities, We can see that

$$P(B|A) = P(B|A^c)$$

Intuitively this suggests independence since the knowledge of A doesn't affect B. We can show this more directly by observing that

$$\begin{aligned} P(B) &= P(A)P(B|A) + P(A^c)P(B|A^c) \\ &= P(A)P(B|A) + P(A^c)P(B|A) \end{aligned}$$

$$\begin{aligned} &= (P(A) + P(A^c))P(B|A) \\ &= P(B|A) \end{aligned}$$

thus A, B independent. It is good to keep in mind the look of a mosaic plot of independent events. It will look as if it is created by a single horizontal and vertical "cut." The cuts could be at any height or width.

The notion of independence applies just as well for random variables. Random variables X and Y are independent if the events $X = x$ and $Y = y$ are indendent for all possible outcomes, x, y. That is, X, Y independent if

$$P(X = x|Y = y) = P(X = x)$$

or, equivalently,

$$P(X = x, Y = y) = P(X = x)P(Y = y)$$

for any choice of outcomes x, y. The intuitive meaning of indepedence for random variables is much like for events — knowledge of the outcome of one random variable does not affect the likelihood of outcomes for the other random variable. For example, if we have a pair of dice with X the first die and Y the second die, these random variables are independent.

1.8.1 Conditional Independence

Now suppose we have two olympic judges who have just seen a performance by a skater and are going to score the performance. Let's write X_1 and X_2 for their rankings, say on a scale of 1 to 10. In modeling X_1, X_2 as random variables we are imagining that the situation could be repeated over and over. In this thought experiment the judges would see the identical performance from the skater, but would have no memory of their early ratings. Thus the judges may give different ratings on different days. Since the judges cannot communicate with each other, the rating of one judge cannot affect the rating of the other, so X_1 and X_2 must be independent.

Now consider a variation on this scenario where we have a *pool* of skaters. We choose a skater S at random from the pool and observe the two judges' scores, X_1, X_2. If we consider X_1, X_2 without knowledge of S, (that is, we look at the marginal distribution of X_1, X_2), then X_1, X_2 would no longer be independent. Some of the skaters are clearly better than others, so when one judge's score is high, the other's will tend to be high too, and vice-versa. This is very much like the example we saw before where we chose a probability p at random and flipped two coins according to that probability. Both the situations of the coins and the skaters can be described accurately with the notion of *conditional independence*. We say that X_1, X_2 are conditionally independent given S. This just means that, restricted to the world where the skater is known, $(S = s)$, X_1 and X_2 are independent:

$$P(X_1 = x_1, X_2 = x_2|S = s) = P(X_1 = x_1|S = s)P(X_2 = x_2|S = s)$$

This is just the usual definition of independence with all three events conditioned on $S = s$. As we have seen above it is possible to have random variables that are conditionally indpendent given some other variable, but not marginally independent (S unknown). We will return to conditional independence in the next section where we talk about naive Bayes classifiers.

Chapter 2

Classification

Now with some probabilistic background we begin looking at *classification*, which is one of the main topics we take up in our course. We first consider the famous iris dataset, which can be accessed in R with

```
data(iris)
```

This dataset describes a collection of 150 iris flowers of three different varieties: *setosa, versicolor, virginica*, with 50 of each. On each of the flowers 4 different features are measured:

1. sepal length
2. sepal width
3. petal length
4. petal width

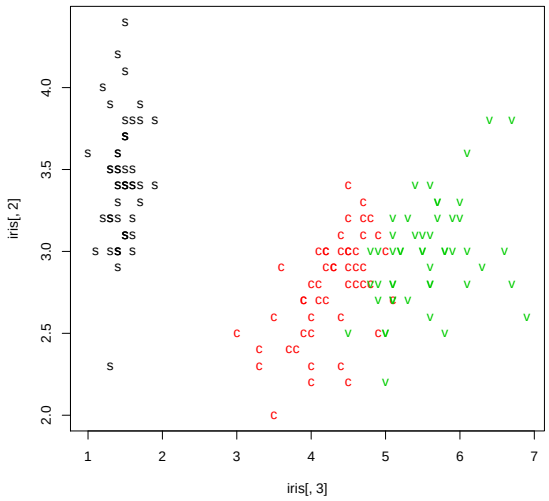
We can think of the data set as an 150×5 matrix with one row for every flower and one column for every feature (as well as an extra column to store the variety of the flower). We will typically write n for the number of instances we have, 150 in this case, so the data matrix could be written as

	sep len	sep wid	pet len	pet wid	class
Flower 1	x_{11}	x_{12}	x_{13}	x_{14}	c_1
Flower 2	x_{21}	x_{22}	x_{23}	x_{24}	c_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Flower n	x_{n1}	x_{n2}	x_{n3}	x_{n4}	c_n

You can easily print out the datset just by typing

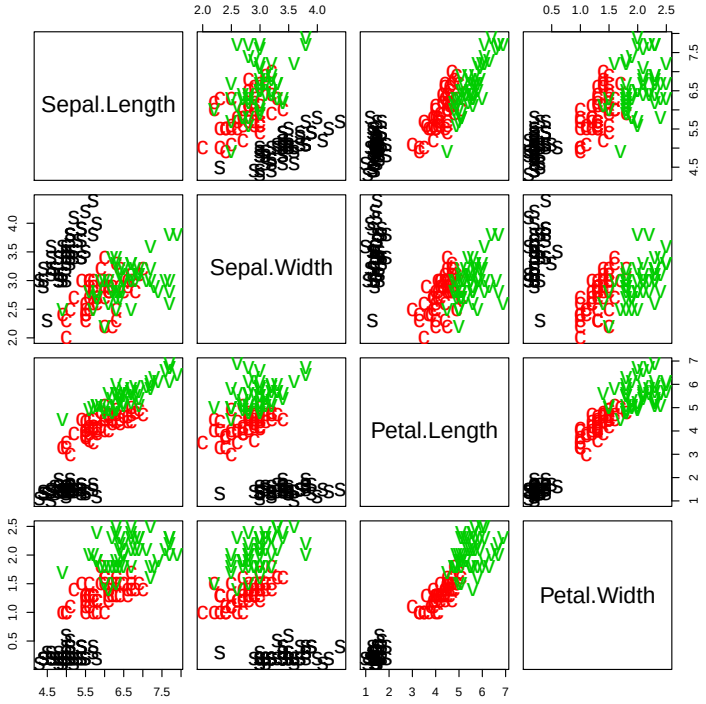
```
iris
```

from the command line, but the raw numbers are hard to interpret. It is worth doing this to see if you can find any patterns of interest. The simplest way of visualizing these data is with a *scatterplot*, which plots two different features as if they were the x-y coordinates of a point. The program iris.r shows a couple of variations on this idea, using different plot symbols and different colors to distinguish the three types of iris. For instance, the figure below shows a scatter plot using two of the features using a different plot symbols (s = setosa, c = versicolor, v = virginica) and color for the three varieties of iris.



From the figure it is easy to see that the setosa iris are easy to distinguish using the chosen features, while there is more ambiguity with the other two varieties.

A scatterplot can only be used with two features since a point in the plane has two coordinates. A generalization of the scatterplot, known as a *pairs plot* shows a scatter plot of each pair of variables. Since we have four variables there are $4 \times 4 = 16$ possible pairs we can make. The pairs plot shows these in a 4×4 matrix of plots with the i, j plot giving a scatterplot of the ith variable (x) plotted against the jth variable (y). The diagonal of this matrix isn't very interesting, since it would plot each variable against itself, so this is usually omitted in a pairs plot in favor of descriptive text. It is worth noting that the i, j element of the pairs plot is just the mirror image of the j, i element, since these both deal with the same two variables, but with opposite use of the two axes. An example is shown below for the iris data.



Using the iris data we could try to figure out which variety of flower we have, only considering the four given features. This is an example of a general kind of problem known as *classification*. In a classification problem we are

given a matrix of feature values along with the true "class" for each instance (row). Using these data we would like to create an algorithm that will assign the class automatically. We hope that our clasification algorithm will do a good job on the examples we are given in the data matrix. But, more importantly, we hope that is will also do well on examples we have not yet seen. That is, we hope our algorithm will *generalize* well.

Without being too mathematical, we could imagine addressing the classification problem visually. Using one of the scatterplots shown in the pairs plot — whichever one we like — we divide the plane into 3 regions with one for each iris variety. Using this approach we would choose the scatter plot where the classes overlap the least, say petal length vs. petal width. With this scatterplot we imagine using two diagonal cuts. The first cut would separate the setosa (black 's') flowers from the other two classes, which can be done perfectly. The second cut would be chosen to separate the versicolor and virginica flowers. Clearly there is no cut that would do this perfectly, but we can get most of them right. Having done this we have separated the plane into three regions. When a new flower (= feature vector) is given to us, we would simply ask which region it lies in and classify by using the label of region.

2.0.1 Classification Examples

Here are a few examples of classification problems:

Spam Suppose we compute a number of features from an email message such as

1. The fraction of capital letters
2. The number of exclamation points
3. A boolean variable asking if the message contains the word "free"

From these feature we would like to classify the message as *Spam* or *Not Spam*.

Poetry We have a newly discovered American poem, and compute a number of features related to rhyme scheme, syllabic scheme, form, syntax, vocabulary, etc. Using these features we wish to determine if the poem was written by *Edgar Allan Poe, Emily Dickinson, Robert Frost, Henry Wadsworth Longfellow*, or *other*.

Mononucleosis Suppose we want to detect whether or not (two classes) a person has mononucleosis. The features we consider are

1. temperature
2. has swolen lymph nodes
3. result of spot test (for antibodies to Epstein-Barr virus).

2.1 The Bayes Classifier

We begin by establishing some notation for the classification problem. We will write $x = (x_1, \ldots, x_p)$ for our feature vector giving the various features we measure. For the rest of this discussion we will ignore the fact that this is a vector, and simply write x, since it keeps the notation simpler. The Bayes' classifier (and most other classifiers) view x as random. For instance, you could imagine simulating the distribution of x by sampling random rows in a data matrix, though one imagines that there is more variability to x that what is exhibited in your data matrix.

Having observed x we want to assign it to a class. We will denote the possible classes as $\{1, 2, \ldots K\}$. We write C for the true class, which is unknown to us. We also regard C as a random variable. One could imagine simulating from the joint distribution of C and x also by sampling the rows of the data matrix, though, again, we would suspect there is variation in these variables wouldn't be completely captured by our data matrix.

Having observed x, *any* class may be possible, but, viewed probabilistically, some are more likely than others. Without worrying about how to do the calculation just yet, we could write

$$P(C = c|x)$$

for the probability of class $c \in \{1, 2, \ldots, K\}$ given x. Here we again follow the notational convention of writing the random variable, C, with a capital letter and its outcome, c, with a small letter. The natural thing to do in this

situation would be to choose the most likely class — this is the choice that has the best chance of being right. This is what the Bayes classifier does.

Said more precisely, if $\hat{C}(x)$ is our *estimate* of the class based on the observed features, x, the Bayes classifier is

$$\hat{C}(x) = \arg\max_c P(C = c|x)$$

The $\arg\max_c$ notation means whichever value c maximizes what follows, so $\hat{C}(x)$ is the class that is most likley having observed x.

2.1.1 Chilean Election Data

The dataset chilean_voting.csv, available in the "Data" directory on Canvas, contains information about the 1988 plebescite in Chile. A plebescite is a *referendum* meaning the entire population can vote. In this case the question was an up or down vote on the president of Chile, Augusto Pinochet, who was removed from the presidency through this vote.

The dataset is a "csv" file, which stands for "comma-separated values." This is a generic data format that can be handled by basically any program that does anything with data. R is no exception. Browsing the file we see that it follows the familiar convention of having one instance (person in this case) per line. Each line is made up of a number of features describing the person, such as age, region, education level, gender, as well as our focus: the vote cast by the person. We view the problem here as one of *classification*, where we seek to predict how a person will vote considering other facts we have at our disposal. You can well imagine the high level of interest in this subject today, and the availability of much richer datasets containing features used for vote classification.

We can read the data in using the R command

```
X = read.csv2("data/chilean_voting.csv",stringsAsFactors=FALSE, sep=",")
```

as done in the chilean_voting.r example. Be sure to tell R the right place to look for the data file. After this command is executed the R object, X, is matrix with the usual rows and columns, one row for each instance, one column for each feature. This data set contains some problems that are frequently encountered in real-world data. Thus we begin with a "data cleaning" phase to simplify things through the commands below.

```
                                                # clean data to retain only yes/no no voters
vote = 9;                                       # the vote (Y,N,A,U) is 9th column
yes = (X[,vote] == "Y" & !(is.na(X[,vote])))    # boolean vector giving "Yes" voters who are not missing
no = (X[,vote] == "N" & !(is.na(X[,vote])))
X = X[no | yes,]                                # keep only the rows with Y/N votes.
```

The 9th column of our data set is the one giving the vote. Ideally we would just like to have two values for the vote, Y/N, but there were also abstentions (A) and undecided people (U), as well as missing values, which we remove from our data set for simplcity's sake. The variable yes in the code is a boolean vector giving TRUE for people who were Y voters (and were not missing). Similary for no. Thus the last statment retains only rows of X corresponding to Y/N voters.

We will build several Bayes classifiers using the cleaned data, using progressively more of the features at our disposal. We begin with the most simple-minded possible approach, as follows.

The R command table tabulates data, counting the number of times each possible configuration occurs. So, for instance,

```
table(x[,vote])
```

would produce the two-value table that counts the number of yes and no voters shown below

```
  N   Y
889 868
         vote
```

showing that the Y/N voters are split more or less evenly.

Of course, this isn't useful for making a classifier since we need to connect the voting patterns with the features we have measured. The following R code below produces a table on education level and vote

```
education = 6                      # education is the 6th feature in X
t = table(X[,c(education,vote)])
print(t)
```

```
education   N   Y
       P  266 422
       PS 224 130
       S  397 311
      vote
```

For the education variable the three possible values are P = primary, S = secondary, PS = post-seconondary. If we were to use *only* the education variable for classification it would be simple to compute the Bayes classifier. For instance,

$$\begin{aligned}\hat{P}(Y|P) &= \frac{422}{266+422} = .61\\ \hat{P}(N|P) &= \frac{266}{266+422} = .39\end{aligned}$$

so we see that the more liklely vote for a primary-educated voter would be Y. Similarly, for a secondary-educated voter

$$\begin{aligned}\hat{P}(Y|S) &= \frac{311}{397+311} = .44\\ \hat{P}(N|S) &= \frac{397}{397+311} = .56\end{aligned}$$

so the more likely vote for a secondary-educated voter would be N. Finally for a post-secondary-educated voter

$$\begin{aligned}\hat{P}(Y|PS) &= \frac{130}{224+130} = .37\\ \hat{P}(N|PS) &= \frac{224}{224+130} = .63\end{aligned}$$

so the more likely vote would also be N. Since the Bayes classifer chooses the most likely vote class conditioned on the data (education level here), we could summarize the classifier in the following table:

		Vote		
		N	Y	$\hat{C}$
Education	P	266	422	Y
	PS	224	130	N
	S	397	311	N

Clearly the class with the greater probability is just the one with more voters.

Similar tables could be constructed using the gender and the region variable where the possible regions are C = Central, M = Metropolitan Santiago, N = North, S = South, SA = city of Santiago. These are given below with the associated classifiers.

		Vote		
		N	Y	$\hat{C}$
Region	C	210	174	N
	M	18	38	Y
	N	102	135	Y
	S	214	275	Y
	SA	345	246	N

		Vote		
		N	Y	$\hat{C}$
Gender	F	363	480	Y
	M	526	388	N

Of course it seems foolish to hope to classify a person's voting choice based on only a single variable. From the tables we can see that we should not expect our classifiers to be very accurate. For instance, considering the classifier based on region, we see that $P(Y|P) = .61$ so we classified a primary-educated voter as a Y. Our classifier would classify every primary-educated voter as Y, so we should expect to make an error on a primary-educated voter with probability 1-.61 = .39. Clearly this isn't very good. To improve our classifier we create a three-way table using both education and gender as "predictors" for the vote. This R table would be created by

```
t = table(X[,c(education,gender,vote)])
print(t)
```

and is summarized below

		Vote		
		N	Y	$\hat{C}$
(Education,Gender)	(P,M)	154	172	Y
	(P,F)	112	250	Y
	(S,M)	234	145	N
	(S,F)	163	166	Y
	(PS,M)	138	170	N
	(PS,F)	86	60	N

Or, using gender, education, and region, the possible outcomes of our feature vector would be

$$\{P, S, PS\} \times \{N, C, S, SA, M\} \times \{M, F\}$$

for a total of $3 \times 5 \times 2 = 30$ rows. We won't list the entire table, but it would begin as below:

		Vote		
		N	Y	$\hat{C}$
(Gender,Education,Region)	(M,P,N)	15	27	Y
	(M,P,C)	49	34	N
	(M,P,S)	384	69	N
	$\vdots$	$\vdots$	$\vdots$	$\vdots$

If we continued down this path including an age variable, say quantized into decades 2 through 8, and an income variable say quantized into the quartiles, Q1,Q2,Q3,Q4 then the possible observations for our feature vector become

$$\{P, S, PS\} \times \{N, C, S, SA, M\} \times \{M, F\} \times \{2, 3, 4, 5, 6, 7, 8\} \times \{Q1, Q2, Q3, Q4\}$$

for a total of $3 \times 5 \times 2 \times 7 \times 4 = 840$ possibilities. However, the number of observations we have remains constant at $n = 1757$. Thus the counts for the cells in our table get smaller and smaller as we chop up the data set into finer and finer pieces. In fact, we should expect that many, and eventually *most* of our cells would have 0 counts. How should we classify if we have a configuration with 0 Y votes and 0 N votes? Or, if we observed a configuration with 2 Y votes and 1 N vote, given our discussion about confidence intervals, should we be confident that Y is the better classification? While this straight-forward implementation of the Bayes classifier is simple, it clearly has its limitations.

2.1.2 Bayes' Classifier through Bayes' Rule

We have seen that the Bayes' classifer chooses the most likely class, $\hat{C}(x)$, given the observed feature data:

$$\hat{C}(x) = \arg\max_c P(C = c|x)$$

In some cases we can compute the conditional distribution above directly as we did in the Chilean election example. But it is more common to implement the Bayes' classifier through Bayes' rule for computing conditional probabilities.

To this end, suppose we have a distribution on the class $P(C = c)$ for $c \in \{1, 2, \ldots, K\}$. This is often known as the *prior* distribion, since it tells us which classes are likely *before* we have observed any data. Usually one estimates

the prior distribution as the proporition of each class in the data matrix. For each class, c, we also have a distribution on our feature vector x, $P(x|C = c)$. These are known as the *class-conditional* distributions, since they give the distribution on our feature vector when the class, c, is *known*. One could approximate the class-conditionals by looking at each class separately, and observing how often each possible feature vector occurs.

Assume we have the prior distribution, $P(C = c)$, and the class-conditional distributions, $P(x|C = c)$ we can use Bayes' rule to compute the distribution on the class *given* the feature vector, $P(C = c|x)$. This latter distribution is known as the *posterior* distribution since it is the distribution on the class *after* we have observed the feature vector. That is, using Bayes' rule

$$\begin{aligned} P(C = c|x) &= \frac{P(C = c, x)}{P(x)} \\ &= \frac{P(C = c)P(x|C = c)}{P(C = 1)P(x|C = 1) + P(C = 2)P(x|C = 2) + \ldots + P(C = K)P(x|C = K)} \\ &= \frac{P(C = c)P(x|C = c)}{\sum_{c'} P(C = c')P(x|C = c')} \end{aligned}$$

Using this we see that the Bayes' classifier can be written as

$$\begin{aligned} \hat{C}(x) &= \arg\max_c P(C = c|x) \\ &= \arg\max_c \frac{P(C = c)P(x|C = c)}{P(x)} \\ &= \arg\max_c P(C = c)P(x|C = c) \end{aligned}$$

In other words, since the denominator is a constant (we are maximizing over c while x is fixed) we only need to maximize the numerator in Bayes' rule. In summary, to implement the Bayes' classifier we look for the class that maximizes the class prior *times* the class conditional, as above. So the product $P(C = c)P(x|C = c)$ is a "score" that measures the appropriateness of the class choice. The first factor biases a choice toward more likely classes, while the second factor argues for classes that are consistent with the observed data. The choice made by the Bayes classifier is a tradeoff between these two factors.

2.1.3 Bayes' Classifier on Iris Data

Consider implementing the Bayes' classifier, using the Bayes' rule approach from Section 2.1.2, in the case of the Iris data:

$$\hat{C}(x) = \arg\max_c P(C = c)P(x|C = c)$$

Here x is the observed vector of 4 feature values (sepal width, sepal length, petal width, petal length) and the class $c \in \{\text{setosa}, \text{versicolor}, \text{virginica}\}$. Of course, we do not know the *true* probabilities, but can estimate these from our data by counting.

Estimating the *prior* probabilities is easy — we have $\hat{P}(C = c) = \frac{50}{150} = \frac{1}{3}$ for each of the three classes. One could argue that this isn't a valid estimate since we don't have a random sample from our classes — by *design* we have 50 examples from each, so we really have no way of knowing which of the iris varieties is more or less prevalent. But this is a minor problem we will not worry about.

The real challenge is with estimating the class-conditional distributions $P(x|C = c)$. There are two problems we need to contend with. One is that the feature vector is 4-dimensional: $x = (\text{sepal wid}, \text{sepal len}, \text{petal wid}, \text{petal len})$. We will handle this simply for now, by only using one of the four features, say petal length, though we will return to this problem shortly. The other problem is that the features are continuous-valued — we have not treated continuous probability yet, and will not do so in our class. However, we can handle the situation by "quantizing" or "binning" as we will demonstrate here.

As review, we can always estimate a distribution on a discrete random variable by just counting the proportion of times each outcome occurs from a random sample. For instance, if X is the number of heads acheived in 3 flips of a fair coin, we could flip a collection of 3 coins n times and estimate $P(X = x)$ by

$$\hat{P}(X = x) = \frac{\#X = x}{n}$$

where $x = 0, 1, 2, 3$. This approach is demonstrated in estimate_discrete_dist.r where we simulate the flipping of the coin with the usual bag of tricks.

With a continuous variable, like petal width, we can do something much like this by dividing the possible range of the variable into several bins and computing the observed proportion of petal width measurements that lie in each bin. The R example iris_one_feature_bayes_class.r demonstrates this approach, first calculating the class-conditional distributions for the petal length, and then implementing the Bayes' classifier.

There is a remaining challenge we postponed discussing. In the implementation just described we chose only one of the 4 features and built our classifier using this feature. Below we consider the cases involving using 1,2, and 4 features, bringing the challenge better into focus. In the examples below we assume that each continuous feature is quantized into 5 different possible values.

1 feature If we have only a single feature, x_1, then we must learn

$$\hat{P}(x_1 = 1, 2, \ldots, 5 | C = 1, 2, 3)$$

for a total of 15 numbers.

2 features If we have two features, x_1, x_2, then we must learn

$$\hat{P}(x_1 = i, x_2 = j, \;\; i, j = 1, 2, \ldots, 5 | C = 1, 2, 3)$$

for a total of $5 \times 5 \times 3 = 75$ numbers.

4 features For 4 features, x_1, x_2, x_3, x_4 we need

$$\hat{P}(x_1 = i, x_2 = j, x_3 = k, x_4 = l \;\; i, j, k, l = 1, \ldots, 5 | C = 1, 2, 3)$$

for a total of $5 \times 5 \times 5 \times 5 \times 3 = 1875$ numbers.

Keep in mind that the number of learning examples we have $n = 150$ doesn't increase as we consider more and more features. It isn't realistic for us to hope to learn 1875 probabilities from 150 examples, since most of our probabilty estimates will simply be 0. The result will be a classifier that performs poorly on any data other than our 150 original "training" examples.

The problem in which the number of needed probabilities quickly swamps the number of training examples as the features increase is known as the "curse of dimensionality." The necessary probabilities grow exponentially (by a constant factor each time we add a feature), very quickly leading to an intractable learning problem.

2.1.4 Naive Bayes Classifier

As an antidote to the *curse of dimensionality* we can make simplifying assumptions in our model. The most common and well-known simplification is the one made by the Naive Bayes classifier. Naive Bayes assumes that the features are *conditionally independent* given the class. One way of thinking about what this means is to have a model for *generating* data according to the conditional independence assumption, as follows. Suppose we have *marginal* (one-dimensional) distributions for each of the 4 features given each of the three classes:

$$P(x_1 | C = c)$$
$$P(x_2 | C = c)$$
$$P(x_3 | C = c)$$
$$P(x_4 | C = c)$$

The conditional independence assumption means that, given a choice of the class $C = c$, we could sample indpendently from the 4 different marginal distributions to get an observation of $x_1 \ldots, x_4$. Said another way, the assumption is that, given the class, there is no interaction between the variables $x_1, \ldots, x_4$, as if they were rolls of 4 different (weighted) dice.

Said more precisely, the conditional independence assumption is

$$P(x_1, x_2, x_3, x_4 | C = c) = P(x_1 | C = c) P(x_2 | C = c) P(x_3 | C = c) P(x_4 | C = c)$$

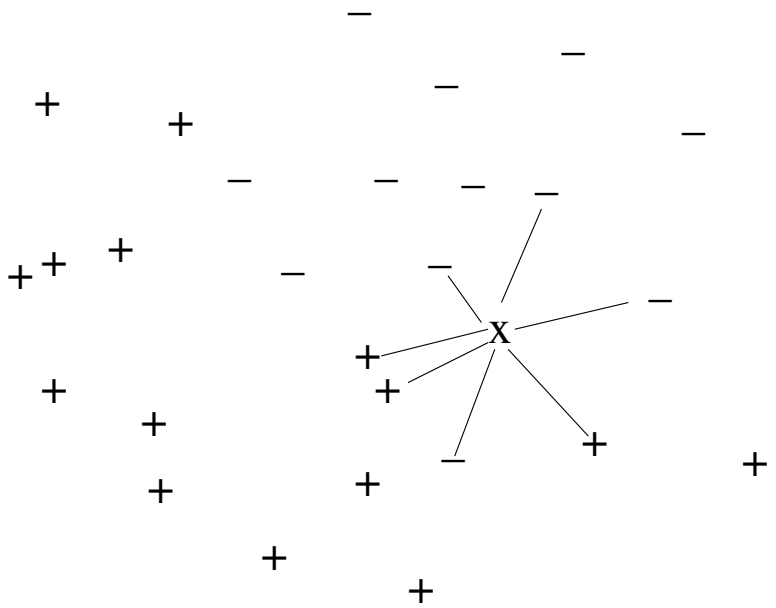
Remember that independence means we can construct the joint from the marginals. That is exactly what we are doing here.

Why is this important? Continuing with our 5-bin quantization, each of the marginal distributions involves $5 \times 3 = 15$ possible probabilites. Since there are 4 different marginals, we need to estimate 60 probabilities. This is *much* better than the 1875 required by the unrestricted joint distribution $P(x_1, x_2, x_3, x_4|C = c)$ described before.

The R program iris_naive_bayes.r shows how to construct this classifier, testing it on the given 150 examples of our dataset.

Nota Bene The Naive Bayes classifier makes an assumption, condtional independence of the features, that is usually wrong. But, even though the assumption is wrong, it may be a good one to make. Broadly speaking, there are two main considerations in choosing a model: *correctness* and *simplicity*. Obviously it is good to have a correct model like the joint model $P(x_1, x_2, x_3, x_4|C = c)$ since it makes no assumtions about the nature of the data — any interactions are possible between the variables. However, the problem with such a model is that complicated models, such as this one, perform poorly in practice. In short, the reason is that the training errors accumulate to create the poor result, though this is part of a larger theme we will return to several times in what is to come. Choosing good models is about finding a good tradeoff between simplicity and correctness. It is often the case that the benefit acheived by moving to the simpler Naive Bayes model greatly outweighs the damage done by the imperfect modeling assumption it makes.

2.2 Nearest Neighbor Classifier



Consider the figure shown above depicting a 2-class classification problem with two continuous features plotted on the x and y axes. The classes are shown with plot characters + and -. In a *nearest neighbor* classifier a point is classified according to the class of the nearest point in the data set. For instance, if we wanted to classify the point marked as x in the figure, we would calculate the distance of this point to the other points, as indicated in the figure. Since the closest "neighbor" is the one in the "10 o'clock" direction, and that point is classified as -, we would classify x as -. Notationally we write $\hat{C}_{NN}(x) = -$.

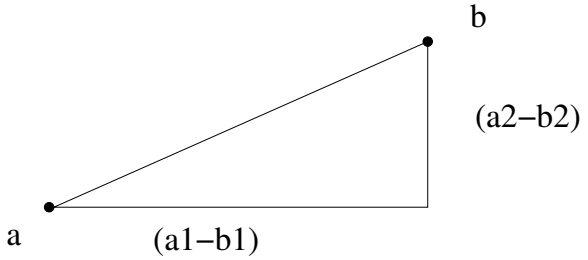
The K-nearest neighbor classifier is a minor varation on this approach. Rather than considering only the single closest neighbor we consider the K closest neighbors. For instance, in the figure above we have identified the 7 closest neighbors. The K-nearest neighbor classifier classifies by voting among the K nearest neighbors. Since 4 of these 7 nearest neighbors are clasified as - while only 3 are classified as + we would classify this point as -. We would write $\hat{C}_{7NN} = -$.

For our discussion to be complete we need to specify the kind of distance we use. There are many possible choices, though the most common is to use Euclidean distance. That is, suppose we have two observations, a and b, and that our observations involve two different features. Thus $a = (a_1, a_2)$ and $b = (b_1, b_2)$. Then the Euclidean distance

between a and b, $d(a,b)$ is given by

$$d(a,b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2}$$

This is just the theorem of Pythagoras used to compute the length of the "hypotenuese."

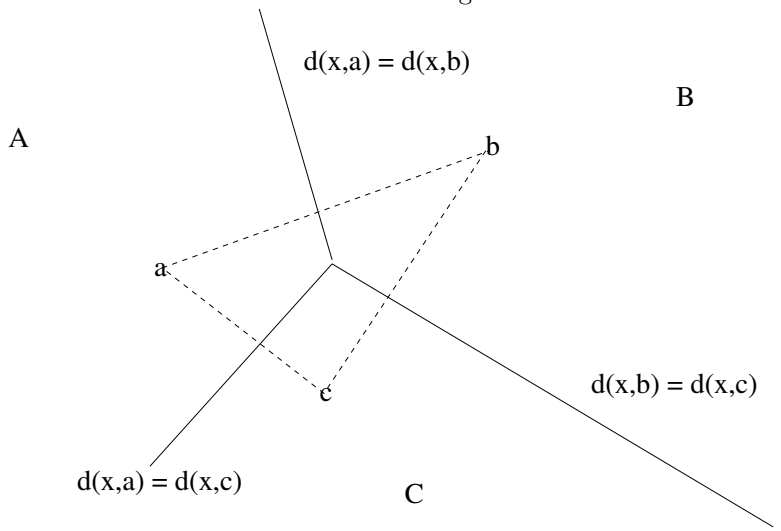


Of course there is no reason we need to confine our use of Euclidean distance to cases where we have only two features. While the two-feature case is easy to draw, we often will have many more features. In the generic p-dimensional case where we have p features, our two instances, a and b will have $a = (a_1, a_2, \ldots, a_p)$ and $b = (b_1, b_2, \ldots, b_p)$. The Euclidean distance generalizes here to

$$\begin{aligned} d(a,b) &= \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \ldots, +(a_p - b_p)^2} \\ &= \left(\sum_{k=1}^{p} (a_k - b_k)^2 \right)^{1/2} \end{aligned}$$

While it is hard to visualize 4-dimensional space, or higher, our notion of Euclidean distance makes complete sense when we are deailing with 4 or more continuous features.

Suppose our data matrix consists of three points, a, b, c with two features for each point. What are the regions in two-space that closest to a? closest to b? closest to c? The figure below shows this situation:



In the figure $\{x : d(x,a) = d(x,b)\}$ is the perpendicular bisector of the segment (a,b), since these are the points, x that are equidistant from a and b. Similarly we can identify the locus of points equidistant from a and c, $\{x : d(x,a) = d(x,c)\}$ and the points equidistant from b and c: $\{x : d(x,b) = d(x,c)\}$. These three perpendicular bisectors intersect in a point that is equidistant from all three points. The result is a division of the plane into three regions, A, B, C where everything in A would be given the same class as a, and similarly for B and C. It is worthwhile to contemplate how this would generalize to more than three points.

2.2.1 Observations on Nearest Neighbor Classifiers

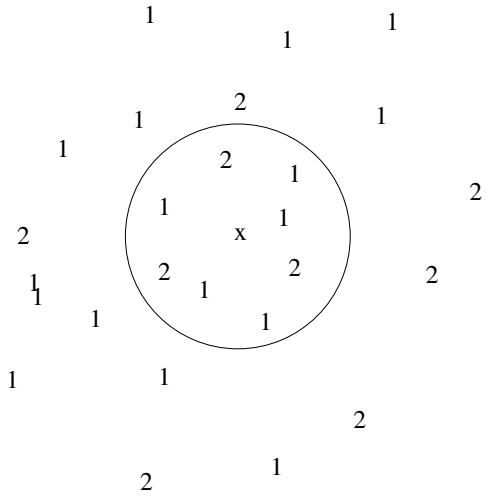
Choice of Distance We discussed using Euclidean distance as the distance measure in constructing the nearest neighbor classifier, but we could substitute any distance we like. For instance, if our data were binary vectors

we could use our our distance "metric"

$$d(a,b) = |\{i : a_i \neq b_i\}|$$

for binary vectors a, b. In choosing a distance metric we hope to find one where vectors separated by small distance tend to have the same class.

Good Results The nearest neighbor classifier is a very simple idea, though it works well in a broad range of practical examples.



Approximates Bayes Classifier Recall that the Bayes classifier classifies a data observation according to the most likely class, *given the data*. We saw that the Bayes classifier is the optimal classifier when the goal is to minimize the error rate. When we looked at the Chilean Voting data, we computed the Bayes classifier simply by counting the Yes and No votes for an observed data configuration, x, assigning the class with the most votes. This was the class with the highest estimated posterior probability.

The K-NN nearest neighbor classifier can be seen as an approximation of this strategy for computing the posterior probability of a class. Consider the figure above where we use a 8-NN classifier to classify the point labeled as x. Here there aren't any data instances that are *exactly* equal to x, so we can't proceed identically with what we did in the Chilean Voting data. But we could approximate the posterior probabilies, $P(C = c|x)$ by counting the class observations in the *neighborhood* of x. In the case depicted we get

$$\begin{aligned} \hat{P}(C = 1|x) &= \frac{\#\text{ 1's near } x}{\#\text{ pts near } x} = \frac{5}{8} \\ \hat{P}(C = 2|x) &= \frac{\#\text{ 2's near } x}{\#\text{ pts near } x} = \frac{3}{8} \end{aligned}$$

Since $\hat{P}(C = 1|x) > \hat{P}(C = 2|x)$ we would classify $\hat{C}_{8-NN}(x) = 1$. Thus the K-NN classifier approximates the Bayes classifier.

2.2.2 Nearest Neighbor Classifier on Iris Data

It is easy to implement a nearest neighbor classifier from a coding perspective. This is demonstrated in the R example iris_nearest_neighbor.r. In R, we begin by computing the "distance matrix", D, between our feature vectors through the command

```
D = as.matrix(dist(iris[,1:4]))
```

To explain this command, recall that the 1st through 4th columns of the iris matrix are the four feature values. The dist command (when followed by the as.matrix cast), produces an $n \times n$ matrix ($n = 150$) giving the Euclidean distance matrix betwen the data vectors. That is D[i,j] is the Euclidean distance between the ith row (flower) and

the jth row (flower):

$$\begin{aligned} D[i,j] &= \left((sl[i]-sl[j])^2+(sw[i]-sw[j])^2+(pl[i]-pl[j])^2+(pw[i]-pw[j])^2\right)^{1/2} \\ &= \left(\sum_{k=1}^{4}(\text{iris}[i,k]-\text{iris}[j,k])^2\right)^{1/2} \end{aligned}$$

Since D is a distance matrix we must have

1. $D[i,j] \geq 0$
2. $D[i,j] = D[j,i]$
3. $D[i,i] = 0$

Once we have the distance matrix, D, it is easy to classify the ith flower. We simply look for the flower that is closest and take its class label:

```
n = nrow(iris)
classhat = rep(0,n)
for (i in 1:n) {
  j = which.min(d[i,])   # the index of the closest flower to ith flower
  classhat[i] = iris[j,5]
}
```

When we observe the results we see that we classified *all* of the examples correctly, which seems like good news — or is it?

Reflecting on this algorithm, the which.min(d[i,]) operation will always give i as a result. This is because the ith row has distance 0 with the ith row, so the resulting classification must always be correct. **But this is misleading.** What we really care about is how the classifier performs on *new* data, not how it performs on the data used to construct the classifier.

2.2.3 Training and Test Data

In the nearest neighbor classifier example, above, we got perfect performance when we tested on the iris data. We saw that the nature of the NN classifier *guaranteed* that each data point used in the construction of the classifier would be classified correctly. Thus, when we tested on these data instances we achieved 100% classification accuracy. While this seems encouraging, we would find when we tested on new data the results would be signficantly worse. The NN classifier shows this tendency in the extreme, but many classification strategies exhibit the same general pattern of classifying more accurately on the data used in classifier construction than other data. We introduce some terminology here to frame this discussion.

Training Data The *training data* are the instances (rows of the data matrix) used in *constructing* our classifier. In the case of the NN neighbor classifier these are the instances we consider when computing the distances. In the Bayes and Naive Bayes classifiers, these are the instances used in estimating the required probabilties — both prior probabilities and class-conditional probabilities.

Test Data The *test data* are the instances (rows of the data matrix) we use in *evaluating* the classifier's error rate. In principle these could be the same as the training data, or they could be completely separate. However, we have seen that testing on the training data results in an overly optimistic notion of error rate — that is, an *underestimate* of classification error rate. This underestimation can be severe, as with the NN classifier, or it can be mild with some other classification schemes. The safest thing will always be to completely separate our training and test data. However, this means we have less data to train the classifier, which is unfortunate.

Genereralization Error In evaluating a classifier the aspect we care about most is how the classifier will perform on data that we have *not yet seen*. After all, this is how it will be used in practice. The error rate on these *not-yet-seen* data is known as the *generalization error rate*. This is the standard by which classifiers are almost always judged. We can estimate the generalization error rate by testing on instances not used in the training data.

Returning to the NN classifier, the fair way to test the classification of an instance, x, would be to look for the closest instance *other than* x, classifying according to the label of that closest instance. This strategy is shown in the R example iris_nn_leave_one_out.r.

This strategy could be explained in the context of the "training" and "testing" terminology as follows. Suppose we begin with our data set $\{x_1, x_2, \ldots, x_n\}$ — these are the rows of our data matrix. In testing x_i we construct the NN neighbor classifier by using all of the instances, *other than* x_i: $\{x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n\}$. In other words, when we classify x_i we only consider distances to instances *other than* x_i. In this way, each x_i is tested on a classifier that was constructed without using x_i. Thus, the resulting estimate of error rate will be accurate.

2.2.4 Cross Validation

The strategy just presented in an example of a more general idea for estimating generization error rate known as *cross validation*.

1. In N-fold cross validation we partition our data set into N different pieces $X_1, \ldots, X_N$. Every instance, x_i will be in exactly one of these pieces. The partition is constructed randomly so that there are no systematic differences between the pieces.

2. For each piece X_i we construct the classifier using the other pieces: $X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_N$. We then test this classifier on the instances in X_i. Thus we ensure that there has been no mixing of training data and test data in the construction and evaluation of the classifier.

3. We estimate the generalization error rate by accumulating the errors the occured over each piece X_i. Our generalization error rate estimate is simply $\frac{\#\text{ Errors}}{n}$ where n is the total number of instances in our data set and # Errors is the number of accumulated errors over all pieces.

4. Having estimated the generalization error rate, we conclude by constructing the classifier using *all* of the available data. This is the classifier we will use for the *not-yet-seen* data. Cross validation is just used as a means of estimating the generalization error rate, not producing the final classifier.

The strategy we employed with the NN-neighbor example is an example of cross validation where, given a data set of n instances, $x_1, \ldots, x_n$ we partition it into n pieces with *one* instance in each piece. For each x_i we create the classifier using all of the other samples (pieces), and test it on x_i. Sometimes this strategy is known as *leave-one-one* cross validation, since each classifier is constructed by leaving one instance out, then testing on that "left out" instance.

2.3 Tree-Structured Classifiers

Among the most popular and versatile classifiers are tree-structured classifiers. This idea developed independently in both the statistics community and the computer science community, thus is often known by one of two names:

1. CART = Classification and Regression Trees

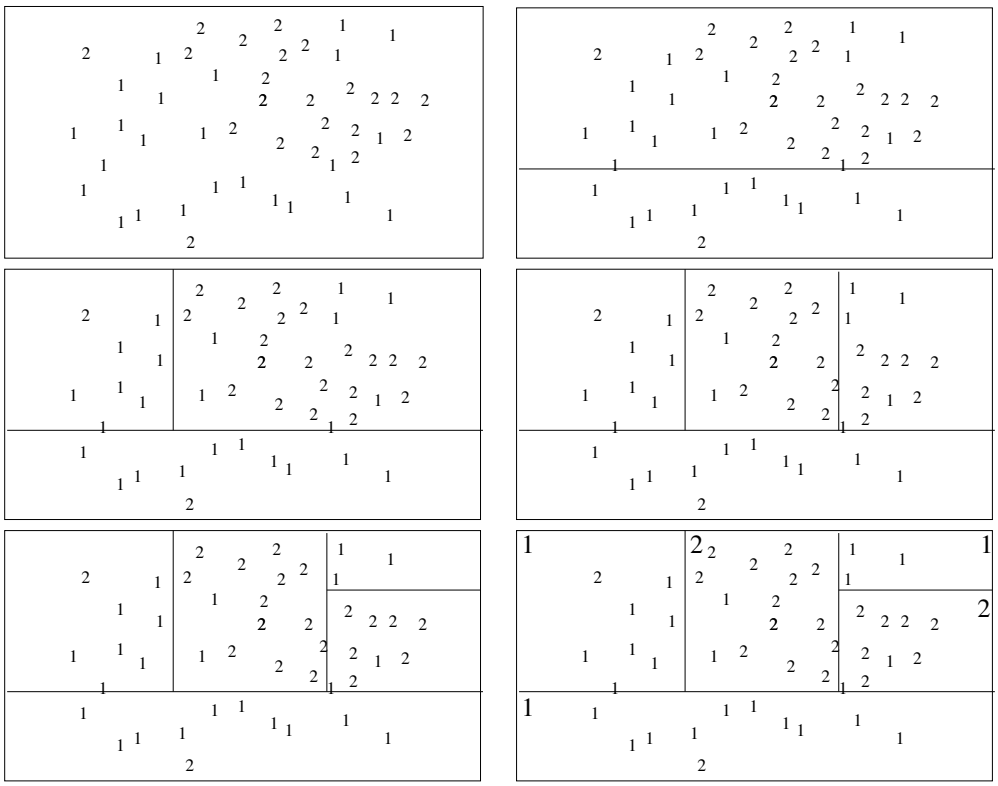
2. C4.5 =

The idea is to build a classifier by recursively dividing up the feature space into smaller and smaller region, splitting on one feature at a time. The idea is best explained through example, so consider the two-class classification data set depicting in the upper left figure below. Consider the following proecure for building theclassifier, illustrated in the figures below.

1. A cut is made by choosing a feature and a split pont. Since we have only two features in this case we must choose either the horizontal or vertical feature. The split point can be any possible value the chosen feature may take. Choose the feature and split point so that the two resulting subregions are as "pure" as possible with regard to class label. Such a split can be visualized by dividing the feature space into two subregions using a horizontal or vertical line, as in the upper-right figure below. Here we choose the make a horizontal cut.

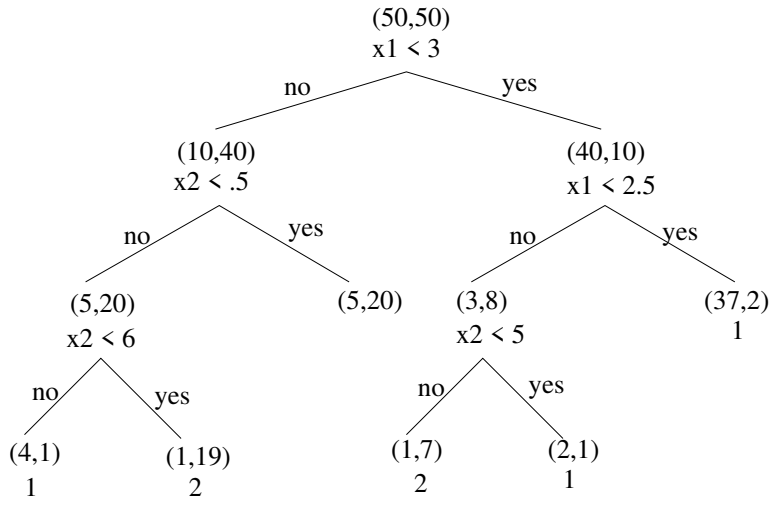
2. Recursively, and "greedily" subdivide the resulting regions to increase the label purity until we decide it is time to stop. (We will discuss stopping later). For instance, the middle-left figure shows a vertical split on the upper rectangle, followed by another vertical split split on the upper-right rectangle, followed by a horizontal split on the upper-right rectangle.

3. Label each of the final regions according to the most prevalent class. These labels are shown in a larger font in the lower-right figure.



The classifier can be visualized as a tree. At each node of the tree we summarize the distribution of class labels by listing the number of each variety we have. So, if we write (a, b) at a node, that would mean that a 1's and b 2's "arrive" at that node. If there are K possible classes we would have K counts in the parentheses. If the node is "terminal," meaning we do not split further, we will write the class label associated with the node — either 1 or 2 in this example. If a node is not terminal the figure describes the split that will occur at the node. For instance, at the root of the tree our split is $x_1 < 3$. For a given instance (row of data matrix) if this condition is true the instance moves to the right child; otherwise it moves to the left child. The intance will continue "dropping down" the tree until a terminal node is reached. The resulting classification would be the label of the terminal node.

For instance, consider the instance $x_1 = 2.7, x_2 = 3.1$. Beginning at the root node we see that the first condition $(x_1 < 3)$ is met, so we would take the right branch. Then we see that the next condition, $x_1 < 2.5$ is not met so we would take the left branch. Finally, the condition $(x_2 < 5)$ is true, so we would take the right branch. We arrive at a node that is labeled as 1, so we classify this observation as class 1.



It is worth noting that tree classifiers adapt equally well to ordinal and categorical data. Ordinal variables require no change in the kinds of splits we consider since the values are ordered. For a categorical variable, such as eye color, we may consider splits of the form $x \in \{\text{brown}, \text{green}\}$ giving a subset of possible values for the categorical variable. We will only consider binary splits corresponding to yes/no questions at a node.

The R example tree_prostate_cancer.r gives an introduction to building a tree classifier to predict a person's outcome after a stage-c prostate cancer diagnosis. Our discussion will reference the treatment of classification trees on the Canvas page: "An Introduction to Recursive Partitioning Using the RPART Routines" by Terry Therneau and Elizabeth Atkinson. (On the website listed as "tree_structured_intro.pdf") This example uses the stagec dataset which is built into the rpart library.

To use tree library, which performs various tasks related to tree-construction, we first must install the rpart package. A package is an "add-on" to the core of R, and is only available once the package has been installed. You can install the package with

```
install.packages("rpart")
```

on the command line. You only need to do this once, so you should not include this command as part of your R program. To access the contents of the package from R you need to use the library command:

```
library(rpart)
```

which loads the library. This also makes available the stagec dataset, which you can see by typing

```
stagec
```

The features in this data set are

pgtime The time to progression of the disease, or the last follow-up free of progression.

pgstat The status at the last follow-up: 1 = progressed; 0 = not progressed. This will be class variable.

age The age of patient at the time of diagnosis

eet A boolean variable. Did the patient get early endocrine therapy?

ploidy The DNA pattern of cancerous cells: diploid/tetraploid/aneuploid

g2 The percentage of cells in the g2 phase (prior to dividing)

grade the grade of the tumor (1-4)

gleason the Gleason grading of the tumor (3-10)

Unlike with other classification methods that we will code ourselves, we use the built-in capabilities of the rpart package to do our tree construction. The following lines of code create and print the the tree:

```
fit  = rpart(progstat ~  age + eet + g2 + grade + gleason + ploidy,
       data = stagec, method ="class",parms = list(split = 'information'))
print(fit)
```

Th rpart command tells R to create a tree classifier that predicts the class variable progstat, using age, eet, g2, ... one the stagec dataset. The printed tree is shown below:

```
n= 146

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 146 54 No (0.6301370 0.3698630)
   2) grade< 2.5 61  9 No (0.8524590 0.1475410) *
   3) grade>=2.5 85 40 Prog (0.4705882 0.5294118)
     6) g2< 13.2 40 17 No (0.5750000 0.4250000)
      12) ploidy=diploid,tetraploid 31 11 No (0.6451613 0.3548387)
        24) g2>=11.845 7  1 No (0.8571429 0.1428571) *
        25) g2< 11.845 24 10 No (0.5833333 0.4166667)
          50) g2< 11.005 17  5 No (0.7058824 0.2941176) *
          51) g2>=11.005 7  2 Prog (0.2857143 0.7142857) *
      13) ploidy=aneuploid 9  3 Prog (0.3333333 0.6666667) *
     7) g2>=13.2 45 17 Prog (0.3777778 0.6222222)
      14) g2>=17.91 22  8 No (0.6363636 0.3636364)
        28) age>=62.5 15  4 No (0.7333333 0.2666667) *
        29) age< 62.5 7  3 Prog (0.4285714 0.5714286) *
      15) g2< 17.91 23  3 Prog (0.1304348 0.8695652) *
```

Each node in the tree is given a number, k, with the convention that its "child" nodes will be numbered $2k$ and $2k+1$. This numbering scheme is also clarified by the indentation scheme so that nodes at the same level of the tree are indented the same amount. Following a node's number is the split taken to reach the node. For instance, the nodes 2,3, which are the children of the "root" node 1, correspond to the instances where grade < 2.5 and grade $\geq$ 2.5. This is the first split of the tree. The next 3 entries are the number of instances that reach the node, the number that would be incorrectly classified if the tree construction stopped at that node, and the classification of the node. For instance, the root node shows 146 54 No. This means that we begin with 146 instances, and that, if the root had to classify with no more information it would classify as No. In doing so 54 instacnes would be classified incorrectly. The root classifies as No because there are 54 Prog instances and 146-54 = 92 No instances, so the majority are No. The numbers in the parentheses are the estimated class probabilities for each node. For instance, at the root node the classifier estimates

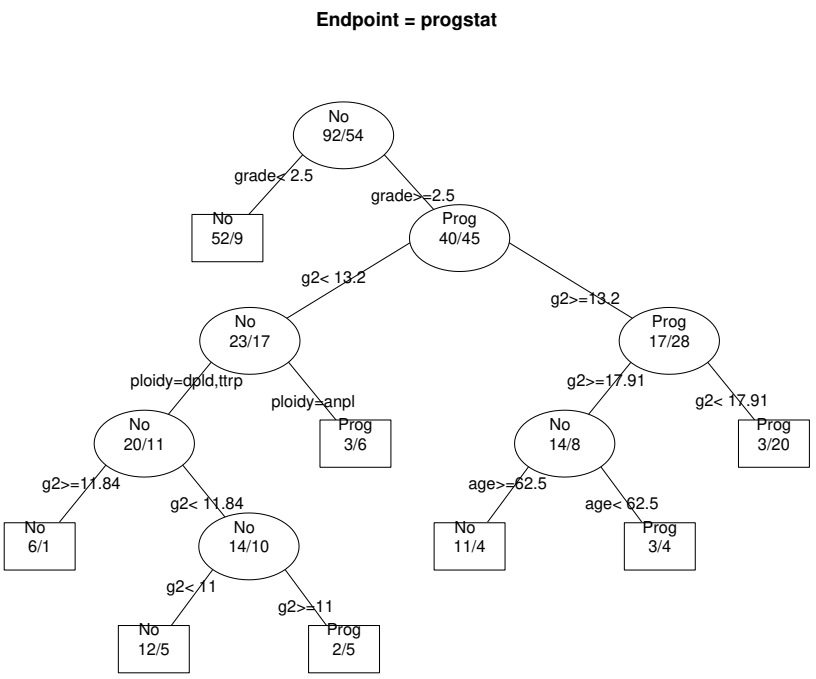
$$\begin{aligned} \hat{P}(C = \text{No}|x) &= \frac{92}{146} = .6301370 \\ \hat{P}(C = \text{Prog}|x) &= \frac{54}{146} = .3698630 \end{aligned}$$

Since this is a probability distribution the numbers in parentheses sum to 1. Finally, the line ends with an asterisk if the node is terminal.

As indicated in the tree_prostate_cancer.r program, R provides a more visual depiction of the tree through

```
post(fit,file="mytree.pdf")
```

This command produces a pdf file (other types of output are possible) that gives the picture of the tree shown below.



Considering this tree, we may want to classify the observation x described by

grade	g2	ploidy	age ...
3	10	tetraploid	70 ...

We begin at the root of the tree where, considering the grade of 3, we take the right branch. There we consider the g2 score of 10, leading to the left branch. The next node splits on the ploidy value, so the value of "tetraploid" brings us to the left branch. Considering again the g2 score we take the right branch followed by the left branch. Since the terminal node is classified as "No" we predict this tumor will not progress.

One may observe from the construction of the tree that the boolean variable eet and the Gleason score are never used in the classification. Tree classifiers are a good candidate when there are many different feature values, since the classifier selects a subset of the feature as the tree is built. Only features that lead to better classification will be used. One may constrast this with other approaches where constructing the classifier becomes harder with more and more features.

2.3.1 Choosing Splits

Recall that the tree is constructed in a *greedy* way, at every branch choosing the feature and split point that maximizes the "purity" of the offspring. There are several ways to make this more precise, though we will only discuss one of them. We will maximum purity by minimizing "entropy."

Entropy

Recall some basic facts about logarithms:

$$\begin{aligned} 2^{10} = 1024 &\quad \Rightarrow \quad \log_2 1024 = 10 \\ 2^0 = 1 &\quad \Rightarrow \quad \log_2 1 = 0 \end{aligned}$$

Suppose we have a probability distribution on K outcomes $p = (p_1, p_2, \ldots, p_K)$. The *entropy* of p, $H(p)$, is defined by

$$\begin{aligned} H(p) &= p_1 \log_2 \frac{1}{p_1} + p_2 \log_2 \frac{1}{p_2} + \ldots + p_K \log_2 \frac{1}{p_K} \\ &= \sum_{k=1}^{K} p_k \log_2 \frac{1}{p_k} \end{aligned}$$

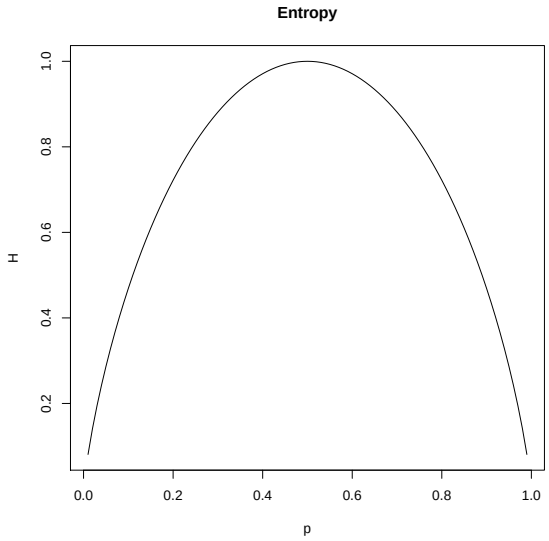
For example, if $p = (\frac{1}{2}, \frac{1}{4}, \frac{1}{4})$,

$$\begin{aligned} H(p) &= \frac{1}{2} \log_2 2 + \frac{1}{4} \log_2 4 + \frac{1}{4} \log_2 4 \\ &= \frac{1}{2}1 + \frac{1}{4}2 + \frac{1}{4}2 = \frac{3}{2} \end{aligned}$$

If one of the probabilities is 0, for the purposes of the entropy calculation we define $0 \log_2 \frac{1}{0} = 0$ since

$$\lim_{p \to 0} p \log_2 \frac{1}{p} = 0$$

For the special case of a two-class problem we have $p = (p_1, p_2)$. In this case we have $p_2 = 1 - p_1$ so we can view the entropy as a function of p_1, as depicted below. We can see the entropy is symmetric, and 0 in the case where $p = (0, 1)$ or $p = (1, 0)$ — that is, the case where there is no real randomness.



What does entropy measure? From the case where $p = (p_1, p_2)$ we see that $H(p) = 0$ when the p is *deterministic* — the outcome of the experiment is known with certainty. On the other hand, we have the *least* knowledge about what will occur in a two-outcome experiment when $p = (\frac{1}{2}, \frac{1}{2})$ which, from the figure, we can see is the case where the entropy is highest. Often people say that entropy measures the "surprise" of a random experiment. Clearly $p = (0, 1)$ has no surprise associated with the outcome, while the maximal surprise will be when $p = (\frac{1}{2}, \frac{1}{2})$, so the interpretation of surprise seems to hold in the two-outcome case. One needs to take a more sophisticated look at entropy than we will to see how it describes the information one gains by observing random experiments. In fact, entropy is the cornerstone of Information Theory. For us, entropy is just the way we will measure the quality of a split — we will seek splits that give class distributions with minimal entropy.

Properties of Entropy

1. $H(p) \geq 0$
2. $H(p) = 0$ only when p "concentrates" on one outcome. That is, $p = (0, \ldots, 0, 1, 0, \ldots, 0)$
3. $H(p)$ is maximal when p has a uniform distribution. That is, $p = (p_1, \ldots, p_K) = (\frac{1}{K}, \ldots, \frac{1}{K})$.

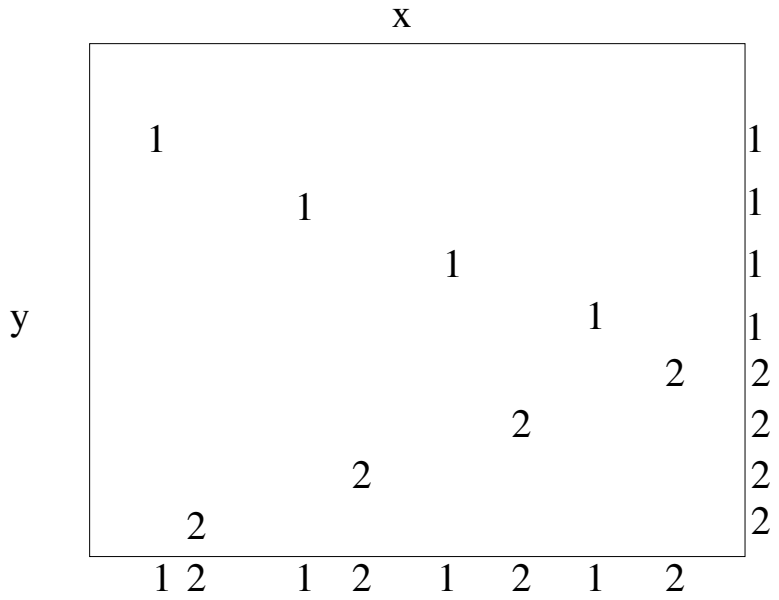
Using Entropy to Choose Split

During the construction of our tree we encounter the same situation over and over. A collection of instances arrive at a node of the tree, and we consider finding a split that best purifies the class distributions of the two child nodes. Now we state this more precisely. Suppose that the instances at the current node have a distribution $p = (p_1, \ldots, p_K)$ on the K possible classes. That is, p_k is the proportion of instances that belong to class k. Any split divides the instances into a "left" group and a "right" group, as they move to the left or right branch of the tree. For the instances that go to the left we write p_L for their class distribution; similarly, we write p_R for the class distribution on instances that go to the right. p_L and p_R both have K components for the K possible classes. Also suppose that q_L is the proportion of instances that go to the left, while q_R is the proportion that go to the right. We will measure the quality of the split by its "average entropy," Q, defined to be

$$Q = q_L H(p_L) + q_R H(p_R)$$

Note that since $q_L + q_R = 1$ this is an average of the left and right class distribution entropies, weighted by the proportions in the left and right groups. We will choose the split that gives the minimal Q — the minimal average entropy.

Let's consider what this does with a specific example. In the figure below we have 8 instances that arrive at a particular node. The instances have two features, labeled as x and y in the figure. The instances belong to one of two different classes, labeled as 1 and 2 in the figure. Before we calculate, decide which variable you think we *should* split on, and where.



First we consider splitting on the x variable — that is, drawing a vertical cut in our rectangle. When we do so, the y values of the instances are not relevant; the instances will be separated solely on the basis of their x values. For this reason we have copied the x values at the bottom of the figure, preserving their x locations.

One possible choice would be to split in the middle of the collection, giving 4 to either side: 1212|1212. In this case $q_L = q_R = \frac{1}{2}$ since half went to the left child and half went to the right child. We also see that $p_L = p_R = (\frac{1}{2}, \frac{1}{2})$ because both the left group and the right group are composed half of 1's and half of 2's. It is easy to see that

$$H(p_L) = H(p_R) = \frac{1}{2}\log_2 2 + \frac{1}{2}\log_2 2 = 1$$

so that the average entropy is

$$Q = \frac{1}{2}1 + \frac{1}{2}1 = 1$$

If we were to split near the edge, say 1|2121212 we would have $q_L = \frac{1}{8}$, $q_R = \frac{7}{8}$, $p_L = (1, 0)$, $p_R = (\frac{3}{7}, \frac{4}{7})$. We have $H(p_L) = 0$ (consider the figure of two-class entropy above), while $H(p_R) = \frac{3}{7}\log_2 \frac{7}{3} + \frac{4}{7}\log_2 \frac{7}{4} = .98$. So

$$Q = \frac{1}{8}(0) + \frac{7}{8}(.98) = .86$$

Both splits give about the same score, and neither lead to very pure children.

On the other hand, suppose we split on the y variable. When we do this the x values are not relevant, so we have lined the y values up on the right margin of the figure above. The obvious place to split here is right in the middle, since it results in completely pure classes. Viewed through the entropy lens, if we split in the middle we get 1111|2222. The left and right class distributions are $(1, 0)$ and $(0, 1)$ both of which have 0 entropy, so the resulting average entropy is

$$Q = \frac{1}{2}(0) + \frac{1}{2}(0) = 0$$

Since this is the smallest the average entropy can be, splitting other places on the y cannot give better (lower) values. Thus we would choose to split on the y value at a point that divides into two equal groups.

In addition to clarifying the meaning of the notation involved in the average entropy, this example shows that minimizing average entropy leads to the choice that makes the most sense intuitively. Thus we will adopt the average entropy criterion for choosing splits in what follows. In the R code for tree construction below, the "split = information" refers to using average entropy (or "information") as a split criterion.

```
fit  = rpart(progstat ~  age + eet + g2 + grade + gleason + ploidy,
       data = stagec, method ="class",parms = list(split = 'information'))
```

2.3.2 Overfitting a Tree

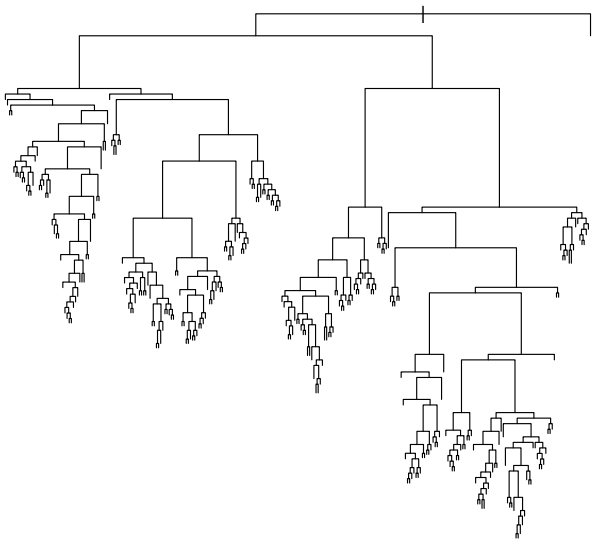
Trees can have many different branches, and thus have an usual degree of flexibility in fitting or explaining a data set. This poses a danger we have seen before where a tree can do a great job of explaining the training data, but classify poorly when we turn out attention to not-yet-seen data.

Here we consider the tree_overfitting.r example on the Canvas page. In this example we create an artificial data set where the features x are *independent* of the class, C. Recall that this means

$$P(C = c) = P(C = c|x)$$

so the opinion we have about the likelihood of the two classes, $P(C = c)$ is *unchanged* after we observe the data x. In other words, the data, x, are of no help in classifying, so we may as well ignore x.

Of course, the tree classifier code in R doesn't know that this problem is hopeless and proceeds doing the usual thing of recursive splitting in an attempt to create nodes with pure class distributions. In constructing the tree we allow nodes to split until there is only a single instance in each terminal node. Normally there is no point in building such a deep tree, though we do it here for illustration's sake. The figure below shows the deep tree that is constructed.

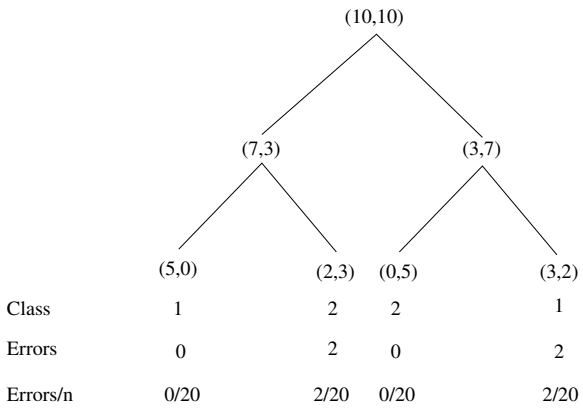


When we test the tree on our training data, knowing by now that this is not a good idea, we get nearly perfect results. With so many branches at its disposal, the tree could perfectly fit the training data. However, we have learned that the important thing is how the classifier behaves on data *like* the training data, but not identical. So we will synthesize some test data from the *identical* model as used in generating the training data. You can compre the code in tree_overfitting.r and see that our test data behave identically to the training. However, when we test on these data we correctly classify only about *half* of the instances. A moment's reflection will show, since there is no relationship between our features and the class, that it is not possible for any classifier to do better than this.

This situation is often referred to as "overfitting" — meaning that we have adapted our classifier to the training data well, but it does not *generalize* well. We will now take up the subject of how to avoid overfitting with trees.

2.3.3 How to Avoid Overfitting the Tree?

Consider the two-class classification tree on classes $\{1, 2\}$ depicted below. In the figure we have written (a, b) at each node, meaning that a 1's and b 2's arrive at the node. We haven't labeled the splits at the node, since they are not important for the current discussion. For each terminal node the figure gives the class associated with the node, the errors that would be made at the node, as well as the errors divided by the total number of instances.



For instance, there are 2 1's and 3 2's that arrive at the 2nd-from-left terminal node, so we would classify that node according to the majority class: 2. The 2 1's that arrive at the node would be incorrectly classified, so we would make 2 errors — the number of errors is just the number in the minority class at the node. We write $R(T)$ for the "error rate" of the tree. In this case we make a total of 4 errors, so

$$R(T) = \frac{\text{\# errors}}{n} = \frac{4}{20}$$

If we wanted to compute the error rate recursively we could define an error rate for each node of the tree, T_i, terminal or not, by

$$R(T_i) = \frac{\text{\# errors below node i}}{n}$$

We can then compute the error rate, $R(T_i)$, recursively by

$$R(T_i) = \begin{cases} \frac{\text{errors at i}}{n} & \text{if } i \text{ is terminal} \\ \sum_{i \to j} R(T_j) & \text{otherwise} \end{cases}$$

The notation may seem complicated at first, but this is nothing very deep. The formula says that we define the error rate at each terminal node T_i simply as the number of errors at that node divided by n. These are the numbers that are labeled in the lowest row in the figure above. For a non-termimal node T_i, such as the three other nodes in the figure, we compute $R(T_i)$ by summing the error rate of its two children. Thus the meaning of the notation $\sum_{i \to j}$ is that we sum over the two children of i using j as the index for these children. So, for instance, the error rate for the left and right children of the root would each be $R(T_i) = \frac{0}{20} + \frac{2}{20} = \frac{2}{20}$, while the error rate at the root would be $R(T_i) = \frac{2}{20} + \frac{2}{20} = \frac{4}{20}$. The recursive calculation seems unnecessarily fancy at this point, but its usefulness will become apparent as we continue our discussion.

It is not a good idea to construct our tree by trying to minimizing the tree's error rate, $R(T)$. Almost every split we make will reduce the number of errors so minimizing $R(T)$ would result in massive overfitting — we would simply split until each child node contains instances from a single class, thus $R(T) = 0$. We have seen that such overfitting leads to a classifier that does not generalize well.

Penalized Fit

We would like to find a principled way of choosing our tree. A common way to do this is the create an objective function that measures the "badness" of a particular tree. We can then pose the search as a minimization problem — we seek the tree that minimizes badness. We might try to minimize $R(T)$ — the error rate of the tree — but we have seen this is not a good idea: the error rate will continually decrease as the tree grows, though the generalization performance of the tree starts to get worse at some point. But a small variation on the idea makes more sense.

For any split penalty $\alpha > 0$, define

$$R_\alpha(T) = R(T) + \alpha|T|$$

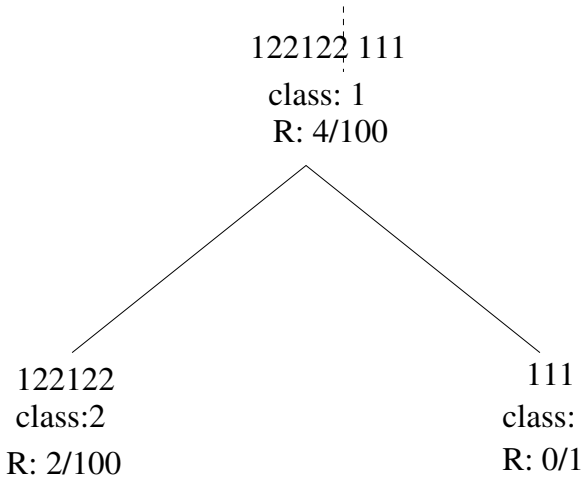
In the above notation $|T|$ is the number of splits in the tree, so $|T| = 3$ in the tree depicted above. Thus $R_\alpha(T)$ includes a penalty of α for each split the tree makes. If we think of minimizing $R_\alpha(T)$ over possible trees, T, the two terms create a kind of "tug of war" as follows:

1. As the tree gets deeper and deeper $R(T)$ gets smaller and smaller, eventually going to 0

2. But, also, as the tree deeper $\alpha|T|$ gets bigger and bigger, increasing by α each time we add a split

So, minimizing $R_\alpha(T)$ gives a compromise between trying to accurately classify our training data and not allowing the tree to get too deep, thus overfitting.

For instance, let's consider the example depicted below where 9 instances arrive at a particular node with $\alpha = .03$. The total number of instances is $n = 100$, in this example.



If we decided *not* to split at the top node in the figure we would label the class as 1 (there are 5 1's and 4 2's), thus making 4 errors, so $R(T_i) = 4/100$. Since the resulting single-node tree has no splits, we also have $R_{\alpha=.03}(T_i) = 4/100$ at the top node. On the other hand, if we choose to split at the indicated position we pay a cost of $\alpha = .03$ for doing so, but manage to decrease the error rate in the left and right children to $2/100$ and $0/100$. Thus splitting creates a tree with score

$$R_\alpha(T) = \frac{2}{100} + \frac{0}{100} + .03$$

In splitting the node our error rate $R(T)$ decreases from $\frac{4}{100}$ to $\frac{2}{100}$, but we also pay a penalty of $\alpha = .03$ for the split so the resulting value of $R_\alpha(T)$ is .05 — according to our *penalized* criterion the tree with the split scores *worse*, so this split is not chosen. We want splits that *decrease* $R_\alpha(T)$.

Computing $R_\alpha(T)$ Recursively

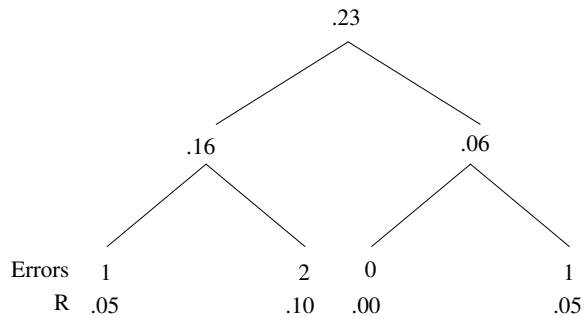
As we did with did with $R(T)$, we can also compute $R_\alpha(T)$ in a recursive manner. In doing this we need to extend our defintion of $R_\alpha(T)$ to any tree node T_i. We do this by

$$R_\alpha(T_i) = \frac{\text{errors in } T_i}{n} + \alpha|T_i|$$

where $|T_i|$ is the number of splits in the subtree below T_i. We now can compute $R_\alpha(T)$ recursively by

$$R_\alpha(T_i) = \begin{cases} R(T_i) & \text{if } i \text{ is terminal} \\ \alpha + \sum_{i \to j} R_\alpha(T_j) & \text{otherwise} \end{cases}$$

Here is an example that illustrates this recursive computation. Suppose we have $n = 20$ instances with $\alpha = .01$, and consider the tree depicted below where we have written $R_\alpha(T_i)$ at each tree node T_i.



At the lowest level of the tree the tree nodes are terminal, so we use the first rule in the above equation for $R_\alpha(T_i)$, just computing the error rate for each terminal, $R(T_i)$. Since there are $n = 20$ examples, the error rate is just the number of errors divided by 20. These numbers are shown under each terminal node. For the next level of the tree

the nodes are no longer terminal, so we use the 2nd rule in the above equation to compute $R_\alpha(T_i)$. So, for instance, for the left child of the root we have

$$R_\alpha(T_i) = \alpha + \sum_{i \to j} R_\alpha(T_j) = .01 + .05 + .10 = .16$$

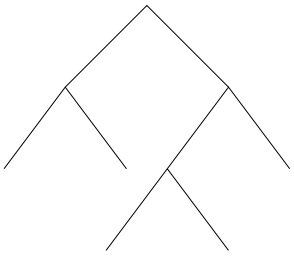
$R_\alpha(T_i)$ can be computed similiary for the other non-terminal tree nodes, leading to $R_\alpha(T) = .23$ at the root. This is the "badness" score we give to the tree.

Alternatively, we could have computed this by the original definition of $R_\alpha(T)$:

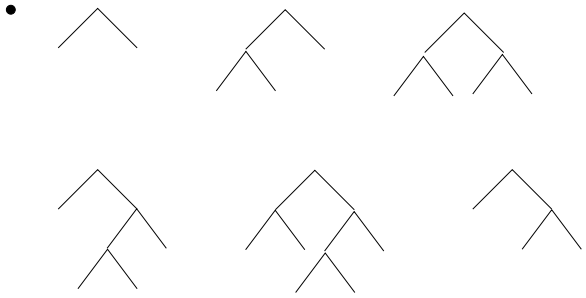
$$R_\alpha(T) = R(T) + \alpha|T| = .20 + 3 \times .01 = .23.$$

Optimizing $R_\alpha(T)$

Suppose we begin by constructing a deep tree — *too deep*. For instance, we could just split each node until we reach a state where the class is completely pure. We have seen that such a tree *overfits* the data, and generalizes poorly — that is, it won't perform well on new data. Now we consider a family of trees known as the *rooted subtrees*. These are all trees that could be created from our original too-deep tree by pruning. That is, by "chopping off" subtrees. For instance, if we consider the original tree shown below



there are 7 possible rooted subtrees including the tree with only a root node:



If we begin with a large tree there will be an *enormous* number of possible rooted subtrees.

Define $R^*_\alpha(T)$ to be the score of the *optimal* rooted subtree. That is

$$R^*_\alpha(T) = \min_{T'} R_\alpha(T')$$

where the minimum is taken over *all* possible rooted subtrees. Even though there are a great many possible rooted subtrees, it is not hard to find the optimal one. To do this we first extend our notion of the optimal tree to subtrees. For any subtree T_i define

$$R^*_\alpha(T_i) = \min_{T'} R_\alpha(T'_i)$$

where the minimum is taken over all possible rooted subtrees of T_i. We can compute $R^*_\alpha(T_i)$ recursively by

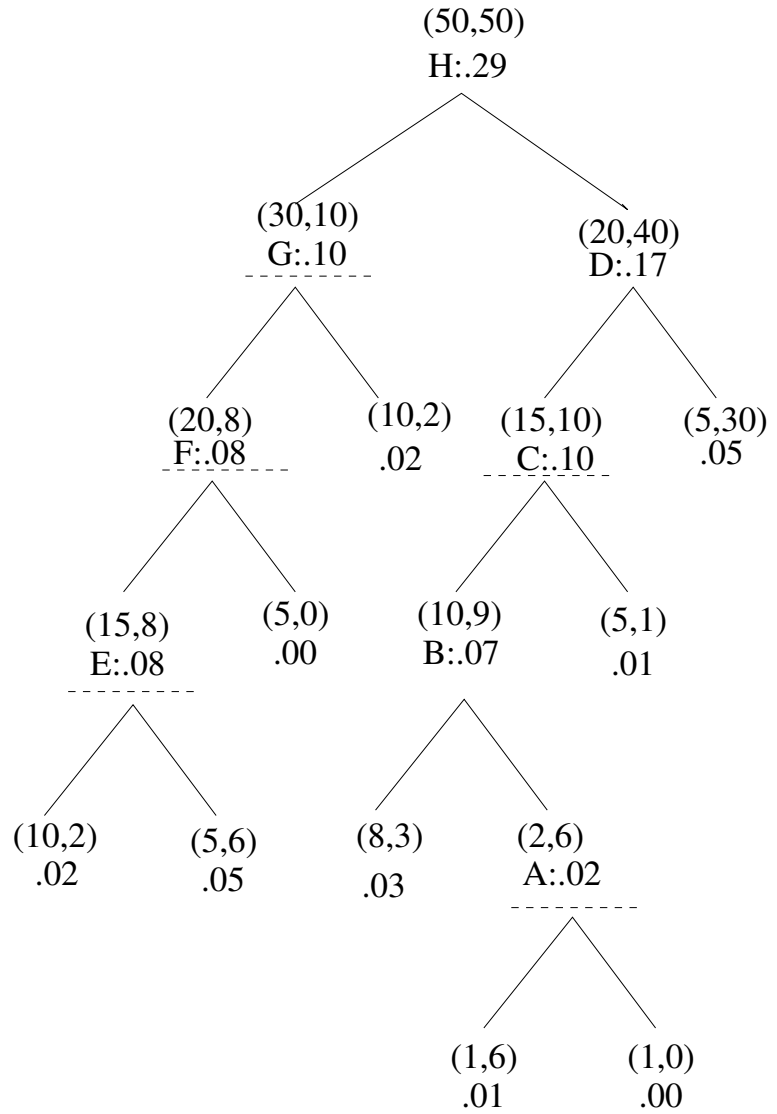
$$R^*_\alpha(T_i) = \begin{cases} R(T_i) & \text{if } i \text{ is terminal} \\ \min(R(T_i), \alpha + \sum_{i \to j} R^*_\alpha(T_j)) & \text{otherwise} \end{cases}$$

In words, this formula considers two possible cases:

1. If a node T_i is a terminal node the optimal score that node can produce is just $R(T_i)$. This is the first line of the formula.
2. For the second line, in computing the optimal score of a non-terminal node, T_i, there are two possible choices:
   (a) We can prune at T_i, in which case the score of the node would be the usual error rate from T_i, $R(T_i)$.
   (b) However, if we choose not to prune then we pay a cost of α for splitting at T_i. The cost due to the left and right subtrees would be from the *optimally-pruned* subtrees we could put there. That is, the costs would be $R^*_\alpha(T_j)$ where j indexes the two subtrees of T_i.

   Having computed the optimal scores due to pruning (a) and not pruning (b), we choose the better (minimum) of these two scores.

While it might not be immediately clear, the recursive formula given above is really an algorithm for computing the optimal rooted subtree and its score. Consider the two-class classification tree depicted below where at each node we write (a,b) where a is the number of examples from class 1 arriving at the node and b is the number of examples from class 2. For this tree we wish to find the best scoring rooted subtree when $\alpha = .02$ — we would get a different result with different choices of the split penalty α.



The formula given above has two cases, terminal nodes and non-terminal nodes, so let's first consider the terminal nodes. If these nodes are contained in our eventual pruned tree, we know exactly how they will contribute to the

total score — we simply record their error rates. We have written these error rates in the figure below each (a,b) pair. For instance, the left most terminal node is labeled (10,2), so would be labeled as class 1, thus making two errors. Given that we have 100 total examples, the error rate is .02, as labeled below this node. The other terminal nodes are labeled similarly.

Now consider the node labeled as A. We have two choices to consider at this node. If we choose to prune we would make 2 errors, thus giving an error rate of .02. If we were to split, we would pay a cost of $\alpha = .02$, while the error rate from the two childen would be .01 and .00. Thus, if we split the score at node A would be .02 + .01 + .00 = .03. Thus pruning gives a better score and this score is .02, as labeled in the figure.

Viewing node A according to our formula we have

$$R^*_\alpha(T_A) = \min(.02, .02 + .01 + .00) = .02$$

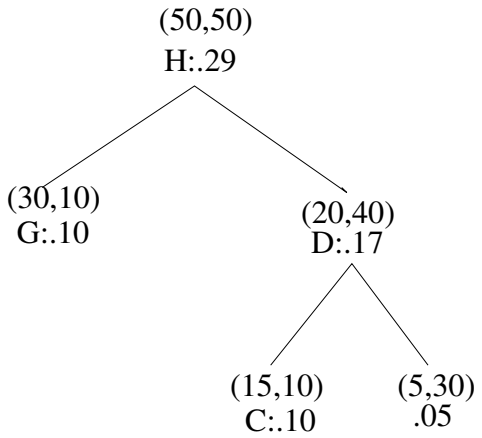
We see that pruning is better than splitting at node A, since the pruning choice led to a lower score. This means if our optimal rooted subtree contains A (it may or may not) we would prune at A. We have indicated this pruning choice with a dashed line at node A.

Now since we have computed the optimal scores associated with the two children of B, we can figure out what the optimal decision would be at node B. If we prune there would be 9 errors so .09 would be our score. If we split we pay a cost of $\alpha = .02$ plus the optimal costs of the two child nodes. Thus the optimal cost for splitting would be .02 + .03 + .02 = .07. This is smaller than the pruning cost, so we would split at node B and incur a cost of .07, as indicated in the figure.

The calculations for all of the nodes are tabulated below. These are computed bottom to top, with the last computation at the root of the tree.

A :	min(.02,.02 + .01 + .00) = .02	prune
B :	min(.02,.02 + .03 + .02) = .07	split
C :	min(.10,.02 + .07 + .01) = .10	no difference
D :	min(.20,.02 + .10 + .05) = .17	split
E :	min(.08,.02 + .02 + .05) = .08	prune
F :	min(.08,.02 + .08 + .00) = .08	prune
G :	min(.10,.02 + .08 + .02) = .10	prune
H :	min(.50,.02 + .10 + .17) = .29	split

Our recursive calcuation shows that the optimal tree would have a score of $R_{\alpha=.02}(T) = .29$. We have marked all of the nodes where the algorithm says to prune with dotted lines, including the "no difference" node, C, where it doesn't matter which choice we make. After pruning our tree in the manner described in the table, the result is the optimal tree shown below.



It is easy to verfify that the $\alpha = .02$ penalized score for this tree is .29.

Choosing α (Cross Validation)

We began by growing a tree that is too deep, and have shown how to compute the optimal rooted subtree for any split penalty α. But how do we choose α. The usual thing to do with "tuning parameters," such as α, is to choose

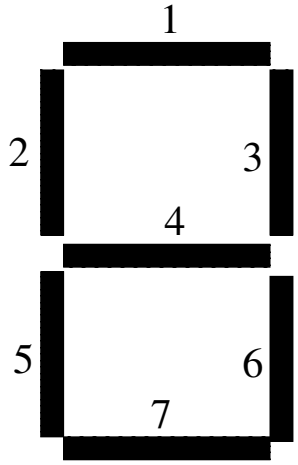
them using cross validation. Here is the method we will use:

1. Choose a collection of candidate split penalties, $\alpha_1, \ldots, \alpha_M$ from smallest to largest. Larger α values correspond to more pruning.
2. Divide the training set randomly into K equal-sized pieces.
3. For each piece $k = 1, \ldots, K$
   (a) Grow a deep tree using all training data *except* the kth piece
   (b) Prune the tree using the candidate penalties $\alpha_1, \ldots, \alpha_M$ resulting in M different trees
   (c) Use these trees to count the number of errors made by instances in the kth data partition
   (d) Accumulate the errors for each value $\alpha_1, \ldots, \alpha_M$ over all data partitions
4. Choose the best penalty α^* to be the one giving the smallest error rate
5. Now using *all* of the training data, construct a deep tree and prune with penalty α^*

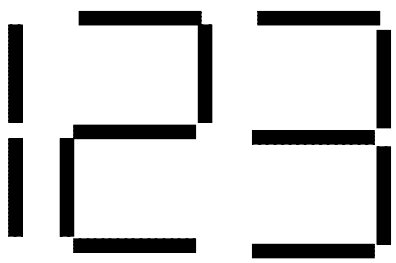
The procedure above is too burdensome to carry out by hand, so we will not do so. The important thing to note is that it is just the usual cross validation scheme for estimating generalization error rate. The only wrinkle is that we apply the process to all of the candidate α values.

An Application: Digit Recognition

The next application uses a synthetic data set from a pioneering book on classification trees by Brieman, Friedman, Olshen, and Stone. The example considers the common LED display often found on appliances, where a decimal digit is created by lighting up a subset of 7 possible display lights. The 7 lights are arranged is in the figure below



For instance we can create the digit one by illuminating light 3 and 6, with different patters for the other digits. For instance:



A table of the light combinations is given below:

	1	2	3	4	5	6	7
0	1	1	1	0	1	1	1
1	0	0	1	0	0	1	0
2	1	0	1	1	1	0	1
3	1	0	1	1	0	1	1
4	0	1	1	1	0	1	0
5	1	1	0	1	0	1	1
6	0	1	0	1	1	1	1
7	1	0	1	0	0	1	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	0

In the classification problem we will address we observe one of the 10 possible sequences of 7 binary digits, but some of the bits will be randomly flipped. For instance, we may observe a version of 2 with the 2nd and 7th digits flipped:

$$2 \rightarrow \underbrace{1011101}_{\text{true bits}} \rightarrow \quad \underbrace{11111100}_{\text{bits 2,7 flipped}}$$

From the "noisy" (bit flipped) bit string we want identify the digit in $\{0, 1, \ldots, 9\}$. So, this is a classification problem with 7 binary features and 10 classes.

We will create a data matrix of $n = 200$ bit strings. While it won't be known to our classifier, the true classes are simply the numbers 0,1,... 9, cycled through 20 times. The first 7 columns of our data matrix are the bit string of the digit with some of the bits randomly flippped. To make the problem more interesting we add 17 more features in columns 8 through 25. These columns just contain random bits. Since the bits in these latter columns are not related to the true class they are useless for classification. Why would we add features that are useless for classification? Part of the challenge in building a classifier is determining which features are valuable and which are not. We should be disappointed to find these latter features used in the classifier since we know they are not useful. So this is a "sanity check" on the classifier's ability to select and use appropriate features.

The situation is summarized in the figure below.

	1 2 ... 7	8 ... 24
1		
2		
3	digit bits with random changes	random bits
4		
⋮		
200		

Our classification tree will be built for us automatically using the rpart package in R, as demonstrated in the digit_recognition.r example. We will discuss this example in some detail here. The example begins with the code

```
library(rpart);
set.seed(1234);  # want random experiments to be exaclty repeatable
n = 200;         # number examples
y = rep(0:9, length = 200);     # the true classes

temp = c(1,1,1,0,1,1,1,         # pattern for 0
         0,0,1,0,0,1,0,         # pattern for 1
         1,0,1,1,1,0,1,         # pattern for 2
         1,0,1,1,0,1,1,         # ...
         0,1,1,1,0,1,1,
         1,1,0,1,0,1,1,
         0,1,0,1,1,1,1,
         1,0,1,0,0,1,0,
```

```
         1,1,1,1,1,1,1,
         1,1,1,1,0,1,0);

lights = matrix(temp,10,7,byrow=T);     # light[i,] is bit pattern for i-1
```

The set.seed(1234) command is a useful programming trick for cases where randomization is used. Randomness in programs can be problematic since debugging requires one to isolate and repeat a problem; the repeatable aspect is lost once randomization is used. However, random numbers on a computer are almost never really random; rather they are "pseudo-random," meaning that they appear to be random, but really are the product of a deterministic algorithm. This fact actually helps us here. By setting the "seed" of the random number generator, we ensure that exactly the same "random" numbers are generated each time the program is run. Thus we will get exactly the same results each time.

In what follows n=200 is the number of examples we generate, while y is 200-length vector containing the true classes, just deterministically cycling through the numbers 0 to 9. temp contains the bit strings associated with each of the digits 0 through 9, while these are collected in the 10×7 matrix lights. Thus lights[i,] is the 7-long bit string for the digit i-1.

The code below constructs the data matrix, x, described by the table above.

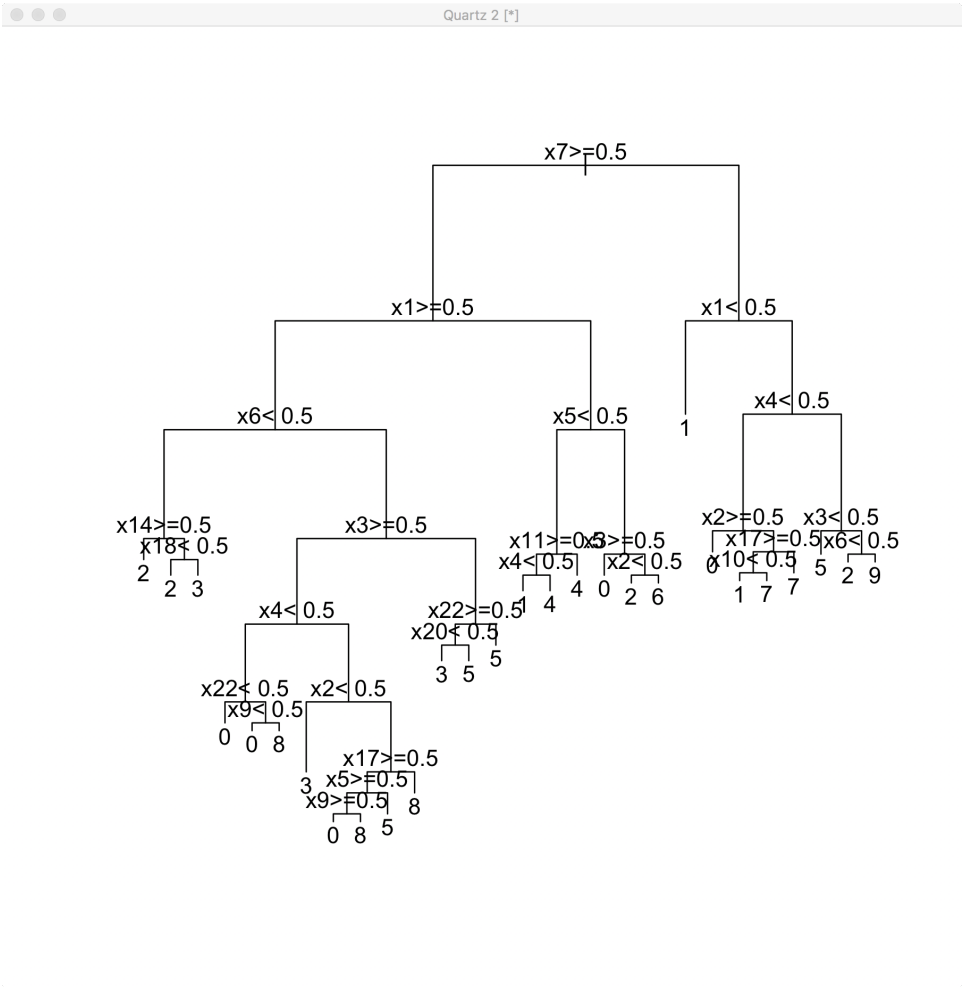
```
temp1 = matrix(rbinom(n*7,1, .9), n, 7)
temp1 = ifelse(lights[y+1,] == 1, temp1, 1-temp1); #  digits with flipped bits
temp2 = matrix(rbinom(n*17,1,.5), n, 17)  # random lights (noisy worthless features)
x = cbind(temp1,temp2);  # variables 8 ... 24 are useless noise
```

temp1 is constructed using some clever R trickery to be the randomized versions of the 200 bit strings. That is the left piece of the table above. These are created so that the probability of flipping a bit is .1 — 10% of the bits are flipped. temp2 is the right portion of the table above, just containing random bits with equal probability of 0 or 1. We concatenate these matrices "horizontally" through cbind to get the matrix described in the table above.

The construction of the classifier is just a single line of code, as below, with several more lines for display.

```
fit = rpart(y ~ x, method = "class", control = rpart.control(xval=10,minbucket=2,cp=0));
plot(fit);
text(fit);    # add text to the tree printout
printcp(fit)  # print out cross-validated error rates
```

Our "fitted" tree is constructed in the first line above, and called fit. This uses the typical R nomenclature where the "tilde" ($\sim$), means to fit or predict the true classes in y using the predictors in x. The fitted tree can be seen in the plot below, which comes from the plot(fit), with text added by the text(fit) command.



R shows us the tree that is produced after the process of growing a too-deep tree. We can see that this tree has overfit the data by the many splits involving variables 8-15 which we know to be useless for classification.

We now take a look at how R prunes the tree, using the ideas we have already discussed. The R command

```
printcp(fit)  # print out cross-validated error rates
```

constructs and prints the table on the complexity parameter (split penalty) shown below

```
          CP nsplit rel error  xerror      xstd
1  0.1111111      0   1.00000 1.10000 0.0078174
2  0.0833333      1   0.88889 1.00556 0.0230371
3  0.0777778      3   0.72222 0.94444 0.0280542
4  0.0750000      4   0.64444 0.81111 0.0348807
5  0.0611111      6   0.49444 0.59444 0.0391873
6  0.0555556      7   0.43333 0.57778 0.0392523
7  0.0500000      8   0.37778 0.53333 0.0392523
8  0.0166667      9   0.32778 0.40556 0.0378247
9  0.0111111     10   0.31111 0.41111 0.0379327
10 0.0055556     11   0.30000 0.41111 0.0379327
11 0.0027778     23   0.23333 0.42778 0.0382305
12 0.0000000     27   0.22222 0.42778 0.0382305
```

The 1st column of this table simply indexes the choice of 12 possible split penalties, shown in the 2nd column. The 3rd column gives the number of splits that would result if we construct the optimal (split-penalized) trees with

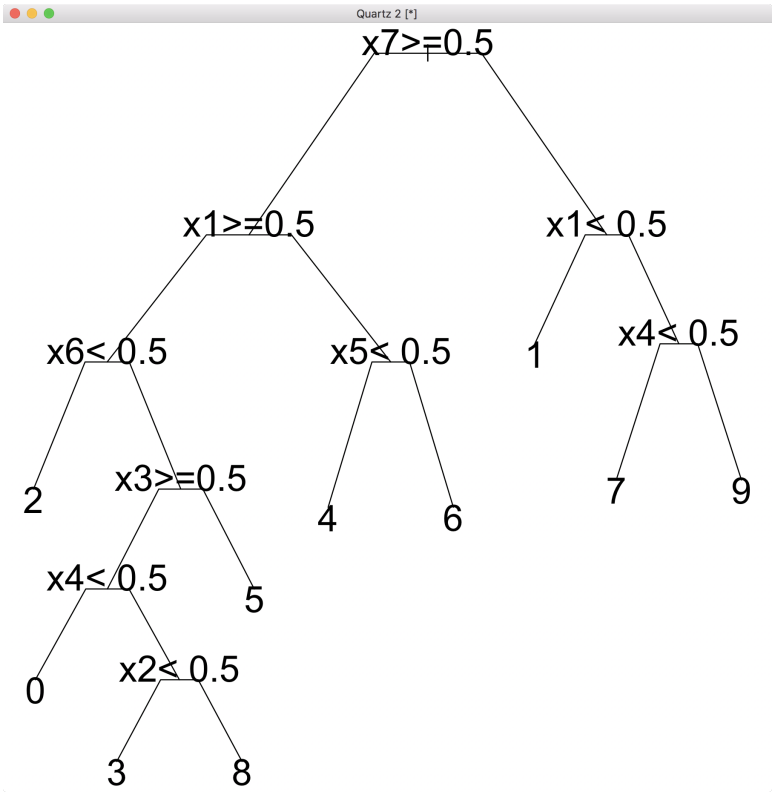
the associated split penalties. For instance, we see that with a split penalty of 0. we construct a very deep tree, as depicted above, while if choose a split penalty of .1111 we would not split at all. The optimal choice will be somewhere in between. The 4th column gives the error rate associated with each tree. This is scaled in an unusual way so that the error rate of the no-split tree will always be 1. In our case, the no split tree would choose a single class for all inputs strings, so it would classify incorrectly 90% of the time. The table calls this .9 error rate a "relative error" of 1.0 and scales all of the other errors similarly. One can see that the error rate given in the 4th column decreases steadily as the tree gets deeper.

The relative error in the 4th column is essentially measuring error on the training set. We have learned that this is not what we should care about. Rather we care about the cross-validated estimate of error where we do not mix training and test. We discussed how to estimate this in the previous section. The cross-validated error is given in the 5th column. Notice that this error does *not* decrease steadily, but rather descrease for a while, and then starts to increase. This is typical of generalization error. Our true error rate (the generalization error) gets better as we add complexity (splits) to the model . . . up to a point. Then the added complexity overfits the model, making the true error rate *worse*. This is exactly what happens in the 5th column.

Of course, we want the choice of complexity parameter that makes the generalization error as small as it can be, and this hapens with a choice of .0166. This would be the value we choose in constructing our final tree. We can produce this tree with the R code

```
fit9 = prune(fit,cp=.017);  # anything in the range .01666 to .05 gives same result
plot(fit9,branch = .3, compress = T);
text(fit9,cex=2);
```

giving the tree below



One can see that only the useful variables (1-7) are used in this tree. This is good news since it means we were able to automatically find a split penalty that resulted in the *correct* choice of variables. (Recall that variables 1-7 are the useful variables while 8-25 are the noise variables).

Chapter 3

Regression

In *classification* we predict a class $C \in \{1, 2, \ldots, K\}$ from a feature vector $x = (x_1, \ldots x_p)$. In *regression* we predict a continuous numerical outcome, y, from a feature vector $x = (x_1, \ldots x_p)$. We now turn our attention to *regression* problems.

Examples

1. Predict the number of people in a particular state infected with COVID-19 tomorrow given the number infected today, the population of the state, and various measures of the amount of interaction between members of the population.

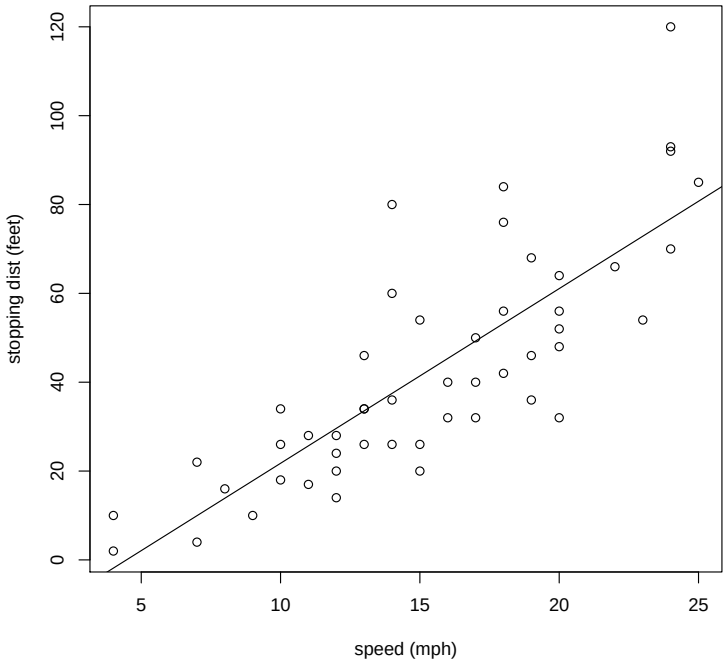
2. Predict a city population 1 year from now considering the current population, the birth rate, the death rate, the immigration rate, the emigraion rate, etc.

3. Predict the life of an electronic part from various measurements.

The most famous special case of regression is *linear* regression, which is what we will study in our class. It is easy to be fooled by the term *linear* since it seems to imply a fairly limited class of models. We will see, in fact, that linear regression includes polynomial regression, regression involving sines and cosines, and a wide variety of possible functional forms. In terms of the "bang for the buck" it is pretty hard to beat linear regression; it is a simple idea that applies to a wide variety of circumstances, and considered to be perhaps the most important general statistical method. (Our class really is a kind of *statistics*).

3.1 Simple Linear Regression

The simplest kind of linear regression is known as *simple linear regression* — this is the classic problem where one fits a line to a collection of two dimensional points. Consider the plot below where we show the stopping times for a collection of cars traveling at different speeds. Each car gives an (x, y) point in the figure below where the x value is the speed of the car (mph) and the y value is the stopping time (feet). Given some arbitrary speed, x, perhaps one not observed in our data set, we would like to predict the stopping time, y. The classic approach to this problem is to "fit a line" to the data, as in the figure below. This line gives us a predicted stopping time, y, for every possible speed, x.



By fitting a line to the data we are assuming that the stopping time is a linear function of the speed. That is, that $y = \alpha x + \beta$ for some unknown constants α and β. Is this a reasonble assumption? From a physical point of it seems that the rate of deceleration (say through wind resistance or skidding on the pavement) would be proportional to the speed. This *non-constant* rate of deceleration isn't consistent with the linear assumption we make. (If you know differential equations you might want to consider what kind of $y(x)$ function *would* be reasonable.) But one could just look at the problem from an empirical point of view — there appears to be a linear trend in the data suggesting that the assumption is okay, at least for the range of speeds considered. We will content ourself with the linear assumption for now, but it is also worth mentioning that the approach we introduce would also work with a more physically-justifiable model.

How to choose α, β? We will construct a measure of error for any given choice of α, β and try to minimize that error over our choice of α, β. Suppose our data are given by $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$. Define

$$E(\alpha, \beta) = \sum_{i=1}^{n} (y_i - (\alpha x_i + \beta))^2$$

The difference $(y_i - (\alpha x_i + \beta))$ measures how much the linear model fails to predict the outcome for the ith data point, so $E(\alpha, \beta)$ measures the overall badness of a particular choice of α, β. It is worth noting that we could have also taken the sum of absolute values, rather than squares, which leads to an equally-plausible objective function, but one that is much harder to optimze.

Finding α, β with Calculus

You likely remember from calculus that we can find the minimum of a function of a single variable by setting the derivative equal to zero and solving for the value of the function. The same idea holds with a function of two variables, like $E(\alpha, \beta)$. Here we set the two *partial* derivatives to 0 — the partial derivatives are obtained by holding one variable fixed and differentiating with respect to the other. We write $\frac{\partial E}{\partial \alpha}, \frac{\partial E}{\partial \beta}$ for the partial derivatives with respect to α and β respectively. In what follows we simply set the partial derivatives equal to 0 and solve for α and β. There is a lot of not-very-important calculation involved, though it is all straightforward, if a bit complicated.

This calculation is included for a couple of reasons. One is just for the sake of completeness, and to show that there isn't any magic involved — just straightforward algebraic manipulation. But there is another point worth

mentioning. Our current framing of the task at hand leads to a rather complicated solution. I still cannot remember these formulas even though I have seen them many times by now. Later we will take a more general view of regression in which simple linear regression is a special case. However, in this more general view, the resulting formulas are much simpler. So we will learn something that is much more widely applicable, and simpler to understand at the same time.

Setting the partial derivatives equal to zero gives

$$\begin{aligned} 0 &= \frac{\partial E}{\partial \beta} = -\sum_{i=1}^{n} 2(y_i - (\alpha x_i + \beta)) \\ 0 &= \frac{\partial E}{\partial \alpha} = -\sum_{i=1}^{n} 2(y_i - (\alpha x_i + \beta))x_i \end{aligned}$$

We can mutiply each equation through by 1/2 to remove the factors of -2, thus leading to

$$\begin{aligned} \sum y_i &= \alpha \sum x_i + n\beta \\ \sum x_i y_i &= \alpha \sum x_i^2 + \beta \sum x_i \end{aligned}$$

Dividing the first equation by $\sum x_i$ and the second equation by $\sum x_i^2$ gives

$$\begin{aligned} \frac{\sum y_i}{\sum x_i} &= \alpha + \frac{n}{\sum x_i}\beta \\ \frac{\sum x_i y_i}{\sum x_i^2} &= \alpha + \frac{\sum x_i}{\sum x_i^2}\beta \end{aligned}$$

and subtracting the 2nd equation from the 1st gives

$$\frac{\sum y_i}{\sum x_i} - \frac{\sum x_i y_i}{\sum x_i^2} = \left(\frac{n}{\sum x_i} - \frac{\sum x_i}{\sum x_i^2}\right)\beta$$

so

$$\begin{aligned} \beta &= \frac{\frac{\sum y_i}{\sum x_i} - \frac{\sum x_i y_i}{\sum x_i^2}}{\frac{n}{\sum x_i} - \frac{\sum x_i}{\sum x_i^2}} \\ &= \frac{\sum y_i \sum x_i^2 - \sum x_i \sum x_i y_i}{n\sum x_i^2 - (\sum x_i)^2} \\ &= \frac{\overline{y}\,\overline{x^2} - \overline{x}\,\overline{xy}}{\overline{x^2} - (\overline{x})^2} \end{aligned}$$

where we introduce the notation

$$\begin{aligned} \overline{x} &= \frac{1}{n}\sum x_i \\ \overline{y} &= \frac{1}{n}\sum y_i \\ \overline{xy} &= \frac{1}{n}\sum x_i y_i \\ \overline{x^2} &= \frac{1}{n}\sum x_i^2 \end{aligned}$$

We can then solve for α by

$$\begin{aligned} \alpha &= \frac{\sum y_i}{\sum x_i} - \frac{n}{\sum x_i}\beta \\ &= \frac{\overline{y}}{\overline{x}} - \frac{1}{\overline{x}}\beta \\ &= \frac{\overline{y} - \beta}{\overline{x}} \end{aligned}$$

Simple linear regression is demonstrated in both the cars_regression.r and simple_regression.r examples on Canvas. The cars_regression.r example is the one that generated the plot above where fit the line to the (x = speed, y = stopping distance) data. It is done by a direct implmentation of the formulas derived above. The simple_regression.r example also uses a direct implementation of the derived formulas, but with synthetic data. In the case we *synthsize* data according to the model

$$y_i = \alpha x_i + \beta + \epsilon_i$$

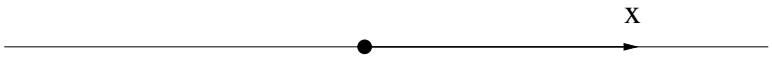
$i = 1, \ldots, n$ where the $\{\epsilon_i\}$ are random numbers with an average or *mean* value of 0. In other words, our $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ data set is created by taking random scatter around a known line. In this example we see that we can recover something very close to the known parameters, α, β using only the (x, y) data points.

3.2 Linear Algebra

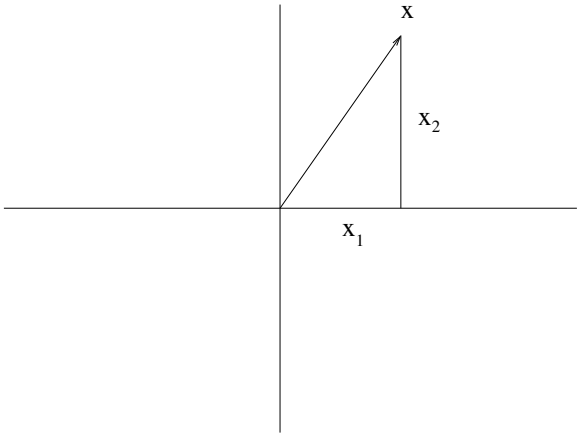
We promised a cleaner and simpler way of doing linear regression, though this requires some background in linear algebra, which is the study of vectors, matrices, etc. We will develop a small amount of this subject, though only a little more than what is needed for the current linear regression discussion.

We write $\Re^p$ for p-dimensional space. For instance

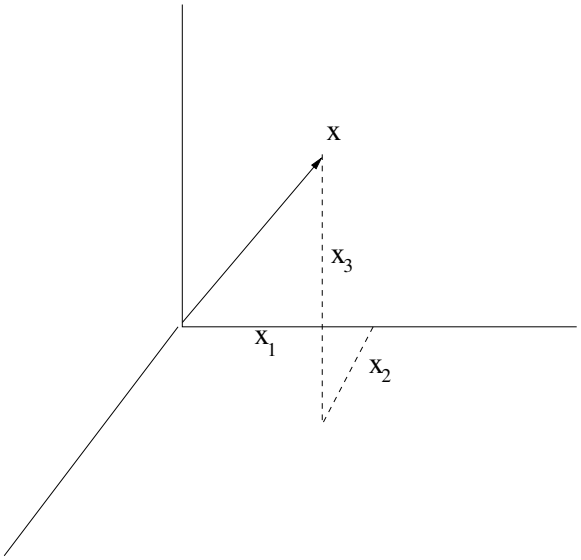
1. $\Re^1 = \Re$ is the collection of real numbers which can be thought of as a line — the "real" line as it is often known.



2. $\Re^2 = \{(x_1, x_2) : x_1, x_2 \in \Re\}$ is the pairs of real numbers, often thought of as a plane since any point in a plane corresponds to the choice of two numbers.



3. $\Re^3 = \{(x_1, x_2, x_3) : x_1, x_2, x_3 \in \Re\}$ is the triples of real numbers, often thought of as 3-space.



More generally $\Re^p$ — p-dimensional space — is the collection of all p-tuples of real numbers. We will typically write something like $x \in \Re^p$ to indicate that x is a p-tuple without necessarily writing out its "coordinates" — the values of the p-tuple. But if we want to be more explicit we write x out as a column:

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{pmatrix}$$

x is often called a "vector," and it can be visualized either as a point or as an arrow emanating from the origin ending at x.

The *length* of a vector x in two-dimensional space ($x \in \Re^2$) can be computed by the Theorem of Pythagoras as

$$\text{Length}(x) = ||x|| = \sqrt{x_1^2 + x_2^2}$$

This generalizes to 3-dimensional x as

$$\text{Length}(x) = ||x|| = \sqrt{x_1^2 + x_2^2 + x_3^2}$$

as one can see from the figure above. More generally we have

$$||x|| = \sqrt{x_1^2 + x_2^2 + \ldots x_p^2} = \sqrt{\sum_{j=1}^{p} x_j^2}$$

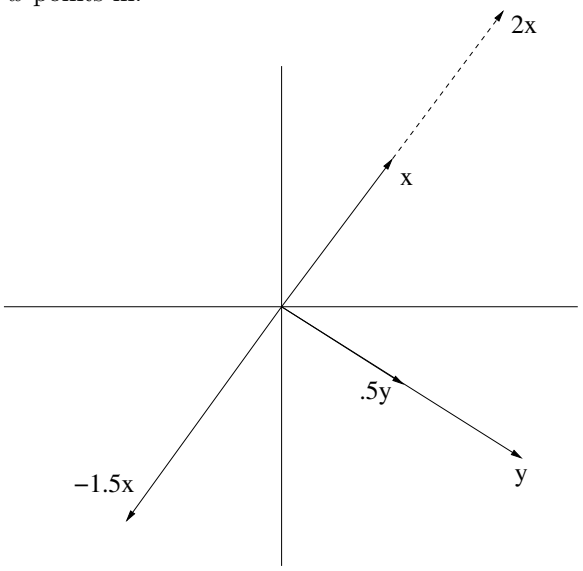
3.2.1 Basic Operations

There are two basic operations that can be performed with vectors. We will define these here as well as note their geometric interpretatons.

1. If $x \in \Re^p$ (x is a p-dimensional vector) and α is a number (sometimes called a "scalar") we define αx to be

$$\alpha x = \begin{pmatrix} \alpha x_1 \\ \alpha x_2 \\ \vdots \\ \alpha x_p \end{pmatrix}$$

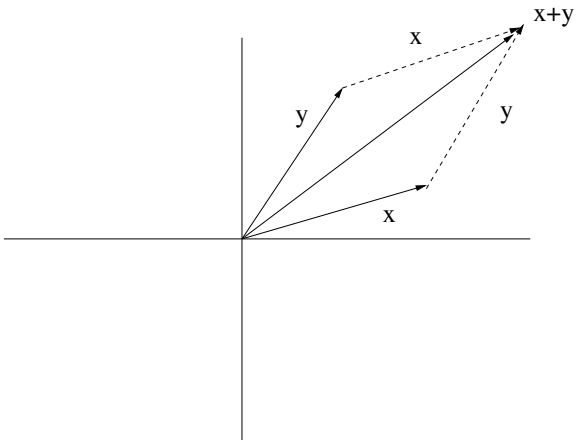
Pictorially, this can be viewed as either stretching x (if $\alpha > 1$) or shortening x (if $0 < \alpha < 1$). If $\alpha < 0$ this would *reverse* the direction x points in.



2. If $x, y \in \Re^p$, $x + y$ is coordinate-wise addition

$$x + y = \begin{pmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \vdots \\ x_p + y_p \end{pmatrix}$$

Pictorially, addition is described by the "parallelogram rule." To add vectors we take one vector and put its origin at the endpoint of the other as in the figure. With vectors $x + y = y + x$ so it doesn't matter which point we start with in the construction. If we do the construction both ways we get a parallelogram, hence the name.



3.2.2 Matrices

A *matrix* is a rectangular array of numbers The matrix is $n \times p$ if it has n rows and p columns. Our data matrix is a good example of such a matrix, though, until know, this has just been a convenient and generic way of storing our data. We will continue to talk about the data matrix in our current discussion, though we will view a matrix not just as a way of storing data, but as an algebraic object that can "do things." If X is a matrix, we can identify its elements with two subscripts, so x_{ij} would be the element in the ith row and the jth column. So we could write out

an $n \times p$ matrix in detail by

$$X = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1p} \\ x_{21} & x_{22} & \dots & x_{2p} \\ \vdots & \vdots & \dots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{np} \end{pmatrix}$$

Multiplying a Vector by a Matrix

If X is an $n \times p$ matrix and a is a p-dimensional vector, we can *multiply* X by a. The result is a n-dimensional vector, y. The elements of y are given by

$$\begin{aligned} y_i &= x_{i1}a_1 + x_{i2}a_2 + \ldots + x_{p1}a_p \\ &= \sum_{j=1}^{p} x_{ij}a_j \end{aligned}$$

$i = 1, 2, \ldots, n$. Pictorially, this could be visualized by

$$i\text{th row} \quad \underbrace{\begin{pmatrix} & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ & & & & & & & & \\ & & & & & & & & \end{pmatrix}}_{X} \underbrace{\begin{pmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \end{pmatrix}}_{a} = \underbrace{\begin{pmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ y_i \\ \cdot \\ \cdot \end{pmatrix}}_{y}$$

As indicated in the figure, the operation $\sum_{j=1}^{p} x_{ij}a_j$ multiplies the ith row of X with a, component-wise, and adds the results together to get the ith component of the result, y. Sometimes this is described by saying y_i is the ith row of X "dotted" with a, or the "dot product" of the ith row of X and y.

Example: System of Linear Equations

Students have likely seen the algebraic problem of solving a system of n linear equations with n unknowns, as below with $n = 2$.

$$\begin{aligned} 3x_1 + 5x_2 &= 7 \\ 2x_1 - 3x_2 &= 4 \end{aligned}$$

Such a system could be written in matrix form as

$$\begin{pmatrix} 3 & 5 \\ 2 & -3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 7 \\ 4 \end{pmatrix}$$

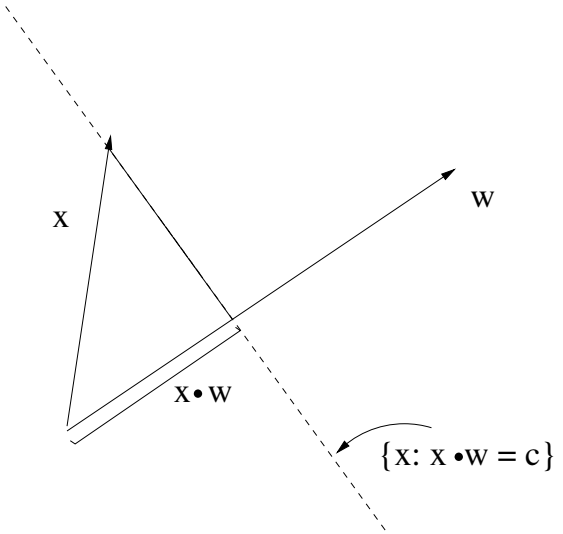
We will discuss how to solve such a system numerically in the coming sections.

Example: Intersecting Lines

The *dot product*, decscribed above, figures prominently in linear algebra, so it deserves a little discussion in its own right. If x and y are p-vectors then the dot product of x and y, is given by

$$\begin{aligned} x \cdot y &= x_1y_1 + x_2y_2 + \ldots + x_py_p \\ &= \sum_{j=1}^{p} x_jy_j \end{aligned}$$

If w is a p-vector of length 1 ($\sum_j w_j^2 = 1$) then $w \cdot x$ is the signed length of the projection of x onto w, as indicated in the figure below. (The signed length would be negative if x pointed in the opposite direction).



If we imagine all of the points in the plane whose projection onto w is some constant, c, $\{x : w \cdot x = c\}$, these points form a line, as indicated in the figure.

Suppose we had two such lines in the plane, described by w_1, c_1 and w_2, c_2 where $w_1 = (w_{11}, w_{12})$ and $w_2 = (w_{21}, w_{22})$, then their intersection, x, would satisfy $w_1 \cdot x = c_1$ *and* $w_2 \cdot x = c_2$, which is equivalent to

$$\underbrace{\begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix}}_{W} \underbrace{\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} c_1 \\ c_2 \end{pmatrix}}_{c}$$

In other words, the point of intersection, x, between the two lines can be found by solving the matrix equation $Wx = c$.

In a similar way if w, x are vectors in 3-space and c is a number, then $\{x : w \cdot x = c\}$ gives a plane. If we had three such planes described by vectors w_1, w_2, w_3 and constants c_1, c_2, c_3, then the point $x = (x_1, x_2, x_3)$ solving

$$\underbrace{\begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{pmatrix}}_{W} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix}}_{c}$$

is the intersection of the 3 planes. This example is one of many interpretations of a matrix equation.

Another Interpretation

Since we know about multiplying vectors by scalars, and adding vectors together, there is an alternative and equivalent description of multiplication of a matrix times a vector. Suppose we write $X_{\cdot j}$ for the jth column of X — this is a little like the R notation

```
x[,j]
```

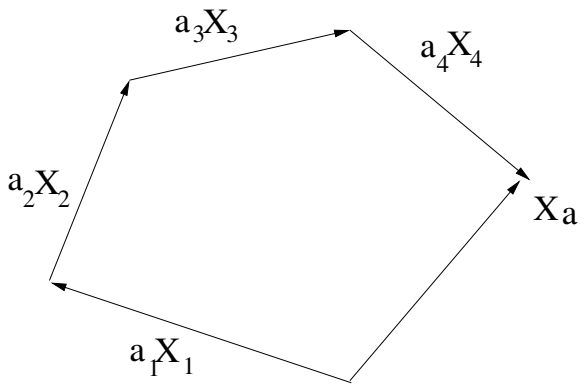
where the missing argument represents *all* values of the argument. In this way we can think of X as

$$X = \begin{pmatrix} \vdots & \vdots & \vdots & \vdots \\ X_{\cdot 1} & X_{\cdot 2} & \dots & X_{\cdot p} \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

Then we can think of the product Xa as

$$Xa = a_1 X_{\cdot 1} + a_2 X_{\cdot 2} + \ldots + a_p X_{\cdot p}$$

In understanding this expression the $a_1, \ldots, a_p$ are just numbers but the $X_{\cdot 1}, \ldots, X_{\cdot p}$ are *columns*. That is, we scale each column of X by the corresponding element of a and add the resulting vectors together. This is indicated pictorially in the figure below.



You should verify for yourself that this is equivalent to the original notion of mutiplication of a matrix times a vector.

Here is an example that shows how one can use this latter interpretation of the product Xa. Suppose we begin at the origin in a plane and are allowed to travel only in 2 directions: (1,2) and (4,1), including the opposites of those two directions. We want to arrive at the point (3,5). How far should we go in each of two directions in order to reach our destination? To solve this, suppose we travel a_1 in the (1,2) direction and a_2 in the (4,1) direction. Then we will arrive at

$$a_1 \begin{pmatrix} 1 \\ 2 \end{pmatrix} + a_2 \begin{pmatrix} 4 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 4 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}$$

That is, we need to solve the matrix equation

$$\begin{pmatrix} 1 & 4 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} 3 \\ 5 \end{pmatrix}$$

for a_1, a_2.

3.2.3 Linear Algebra in R

The last two examples pose real problems as solutions to matrix equations. We will now see how easy it is to solve such equations in R.

We have been using the term vector to describe a collection of numbers in R all along. For instance,

```
x = c(1,2,3)
y = c(4,5,6)
```

creates two R vectors. The operations of multiplying a vector by a scalar and adding two vectors require no special R operations, as we discussed long ago.

```
w = 4*x
z = x+y
```

That is, the first equation performs mutiplication by a scalar, while the 2nd gives the sum of two vectors.

Matrices can be created in several ways, though the easiest is just transform a sequence of numbers into a matrix, as below.

```
X = matrix(c(1,2,3,4,5,6),byrow=T,nrow=2)
```

produces the matrix

```
> X
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

while

```
X = matrix(c(1,2,3,4,5,6),byrow=F,nrow=3)
```

produces the matrix

```
> X
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

If we want to multiply a matrix times a vector, we must be sure the two objects agree — the number of columns in the matrix must be the dimension of the vector. So for instance, the following code

```
X = matrix(c(1,2,3,4,5,6),byrow=F,nrow=3)
a = c(1,2)
y = X %*% a
print(y)
```

produces the output

```
    [,1]
[1,]    9
[2,]   12
[3,]   15
```

Note that we use the new R symbol of the

```
%*%
```

to denote matrix multiplication. So, considering the problem presented above, if we wanted to know how far to travel in the $\begin{pmatrix} 1 \\ 2 \end{pmatrix}$ and $\begin{pmatrix} 4 \\ 1 \end{pmatrix}$ directions to reach the point $\begin{pmatrix} 3 \\ 5 \end{pmatrix}$, we saw that we needed to solve the matrix equation at the end of the previous section. In R, this can be done with the solve command, as follows:

```
X = matrix(c(1,2,4,1),byrow=F,nrow=2)
y = c(3,5)
a = solve(X,y)
print(a)
```

giving the output

```
[1] 2.4285714 0.1428571
```

One can verify that the matrix equation was solved correctly by

```
> print(X %*% a)
     [,1]
[1,]    3
[2,]    5
```

That is, if we go 2.4285714 in the $\begin{pmatrix} 1 \\ 2 \end{pmatrix}$ direction and then 0.1428571 in the $\begin{pmatrix} 4 \\ 1 \end{pmatrix}$ direction we arrive at $\begin{pmatrix} 3 \\ 5 \end{pmatrix}$.

3.2.4 In Class Practice

Here are a few problems we'll go over together to reinforce the ideas presented so far.

1. Suppose we have the system of 2 linear equations in 2 unknowns, x_1, x_2:

$$\begin{array}{rcrcl} 2x_1 & + & 3x_2 & = & 4 \\ 5x_1 & + & 4x_2 & = & 7 \end{array}$$

   Solve for x_1, x_2 using R.

2. Suppose we have three planes given by the equations

$$\begin{array}{rcrcrcl} 1x_1 & + & 2x_2 & + & 3x_3 & = & 0 \\ 2x_1 & + & 3x_2 & + & 4x_3 & = & 1 \\ 1x_1 & - & 1x_2 & + & 1x_3 & = & 2 \end{array}$$

   Find the point of intersection between the 3 planes.

3. Suppose we start at the origin and can travel as far as we want in 3 directions in 3-space. We can travel a negative amount in a direction too. The directions are $(1, 2, 3)$, $(1, -1, 1)$, and $(5, 0, 0)$ How far should we travel in each of these directions to reach the point $(1, 0, 1)$?

4. Three vendors are selling wildflower seeds containing different mixes of poppies, lupine, daisies, and phlox. The table describes the mixes from each of the 4 vendors:

	Poppies	Lupine	Daisies	Phlox
Farmer's market	.7	.2	.1	0.
Felix Flowers	.1	.2	.1	.6
Fleur's Fleurs	.25	.25	.25	.25
Fiora's Fiore	.8	0	.2	.0

   While the arrangement is a little unusual, it is okay to sell seeds back to the vendors using the appropriate mixes. That is, it is okay to buy a negative quantity of seeds from a vendors. How much should be purchased from each vendor so we end up with 3 ounces of poppies, 4 ounces of lupine, 5 ounces of daisies, and 6 ounces of phlox?

3.2.5 Matrix Operations

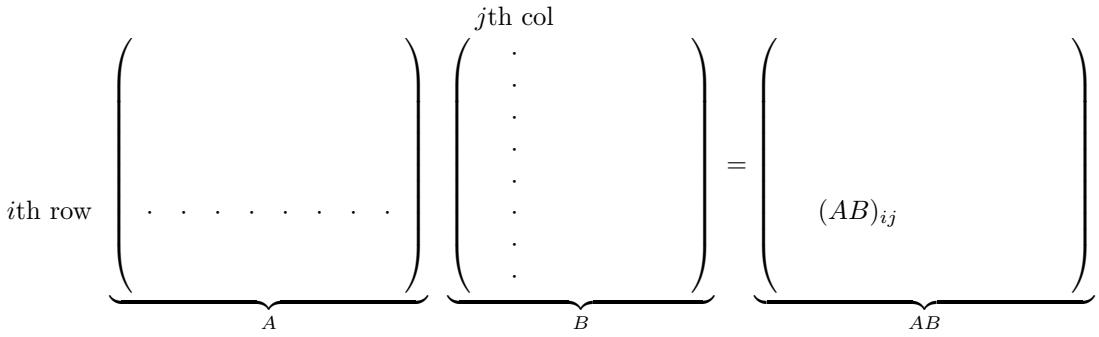
As stated before, there are various things we can "do" with matrices, as follows.

Matrix Multiplication

We can generalize the notion of multiplying a matrix times a vector to multiplying two matrices together. If $\underbrace{A}_{n\times p}$ is an $n \times p$ matrix and $\underbrace{B}_{p\times q}$ is an $p \times q$ matrix (the "middle" dimensions must agree) then we can form the matrix $\underbrace{AB}_{n\times q}$ which will be an $n \times q$ matrix. The i, j element of AB is obtained by "dotting" the ith row of A with the jth column of B

$$\begin{aligned} (AB)_{ij} &= A_{i1}B_{1j} + A_{i2}B_{2j} + \ldots + A_{ip}B_{pj} \\ &= \sum_{k=1}^{p} A_{ik}B_{kj} \end{aligned}$$

Pictorially, this looks like the following:



Writing the dimensions below the matrices for emphasis we get

$$\underbrace{A}_{n\times p}\underbrace{B}_{p\times q} = \underbrace{AB}_{n\times q}$$

If A and B are R matrices we still use the same symbol for matrix multiplication as before

```
A %*% B
```

Transpose of a Matrix

The transpose of an $n \times p$ matrix, A, is a $p \times n$ matrix denoted by A^t. A^t is created by exchanging rows and columns of A — the first row of A is the first *column* of A^t, the 2nd row of A is the 2nd *column* of A^t etc. This is written out more formally by

$$A^t_{ij} = A_{ji}$$

In R we could create a transpose of A by using byrow=F instead of byrow=T.

```
A = matrix(c(1,2,3,4,5,6),byrow=T,nrow=3)
print(A)
B = matrix(c(1,2,3,4,5,6),byrow=F,ncol=3)  # B is the transpose of A
print(B)
B = t(A)    # another way of doing the transpose
print(B)
```

which produces the output

```
>   print(A)
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
>   B = matrix(c(1,2,3,4,5,6),byrow=F,ncol=3)  # B is the transpose of A
>   print(B)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
>   B = t(A)    # another way of doing the transpose
>   print(B)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

The rows of A become the columns of A^t.

3.3 Matrix Formulation of Linear Regression

In *simple* linear regression our data were $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$. The *predictors* $\{x_i\}$ were single numbers and the *responses* $\{y_i\}$ were also single numbers. Here we consider a more general formulation where we have *several* predictor variables $x_{i1}, \ldots, x_{ip}$ for each response y_i. The response remains a single number. We will form our data matrix as we did in classification with one row for each instance and one column for each feature variable (predictor). Thus our matrix of predictor variables X is given by

$$X = \begin{pmatrix} x_{11} & x_{12} & \ldots & x_{1p} \\ x_{21} & x_{22} & \ldots & x_{2p} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \ldots & x_{np} \end{pmatrix}$$

while the vector of responses is

$$y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

In linear regression we seek to predict each y_i from $x_{i1}, \ldots, x_{ip}$. As always, our predictions will be imperfect, so we will write $\hat{y}_i = \hat{y}_i(x_{i1}, \ldots, x_{ip})$ for our prediction of y_i. As with classification, our prediction, *depends* on the predictor variables, or, said another way, is a *function* of the predictor variables — this is the meaning of writing $\hat{y}_i = \hat{y}_i(x_{i1}, \ldots, x_{ip})$. We collect all of the predictions together in a vector $\hat{y}$

$$\hat{y} = \begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_n \end{pmatrix}$$

Linear regression assumes that $\hat{y}_i$ is a *dot product* or *linear combination* of the predictors $x_{i1}, \ldots, x_{ip}$. That is, for some (as yet unknown) $a_1, \ldots, a_p$ we have

$$\hat{y}_i = a_1 x_{i1} + a_2 x_{i2} + \ldots + a_p x_{ip}$$

where we are required to use the same *regression coefficients* $a_1, \ldots, a_p$ for each instance $i = 1, \ldots, n$. Writing this out in detail for all instances, i, we get

$$\begin{array}{ccccccccc} \hat{y}_1 & = & a_1 x_{11} & + & a_2 x_{12} & + & \ldots & + & a_p x_{1p} \\ \hat{y}_2 & = & a_1 x_{21} & + & a_2 x_{22} & + & \ldots & + & a_p x_{2p} \\ \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\ \hat{y}_n & = & a_1 x_{n1} & + & a_2 x_{n2} & + & \ldots & + & a_p x_{np} \end{array}$$

We recognize this as a system of n linear equations with p unknowns. Thus, letting $a = (a_1, \ldots, a_p)$, we can write more compactly as a matrix equation

$$\hat{y} = Xa$$

The Normal Equations

We are now faced with the problem of choosing our regression coefficients, a, so that we get the best prediction of y possible. The criterion we will use is *sum of squared error* — we will choose a so that the resulting sum of squared error (SSE) is as small as it can be. That is, writing $||v||^2 = \sum v_i^2$ for the squared Euclidean distance, we will choose a so that

$$\text{SSE} = \sum_i (y_i - \hat{y}_i)^2 = ||y - \hat{y}||^2 = ||y - Xa||^2$$

is minimized. We choose this criterion because it both makes sense in terms of our objective of good prediction, but also because it is *computationally tractable*, as we shall now see. But how to do this?

We first consider a brief "sidebar" on solving systems of linear equations. We saw when we first introduced systems of linear equations that, barring pathologies, we can solve a system with 2 equations and 2 unknowns. In this case each equation in 2 unknowns can be viewed as a line, so solving 2 such equations would be the intersection of lines – a point. Thus there is a unique solution to a system of 2 equations in 2 unknowns. Similarly, in 3 dimensions a linear equations in 3 unknowns corresponds to a plane, while the itersection of 3 planes is a point. In other words, again barring pathologies, there is a unique solution to a system of 3 equations in 3 unknowns. More generally, once more barring pathologies, there is a unique solution to a system of p equations in p unknowns. But what are these pathologies? The above statements are true if the equation matrices are not *singular*. We will not go into a discussion of the meaning and consequences of singularity, but do want to mention that the story is a little more nuanced than our presentation may suggest.

Returning to our prediction problem, in the best of all possible worlds we would solve the equation $y = Xa$ since then $y - Xa = 0$ (the vector of all 0's) so the resulting SSE is 0. However, we usually cannot do this. X is an $n \times p$ matrix so solving $y = Xa$ is solving a system of n equations in p unknowns $a_1, \ldots, a_p$. Typically we have many more equations than unknowns (n is much greater than p) so this is simply not possible. But a variation on this goal is possible. We will modify the equation $y = Xa$ by multiplying both sides by the matrix X^t, thus giving the *normal equations*

$$\underbrace{X^t y}_{p\times 1} = \underbrace{X^t X}_{p\times p} a$$

If we consider the dimensions of the objects above X^t is $p \times n$ and y is $n \times 1$ — we can think of an n-vector as an $n \times 1$ matrix. So $X^t y$ is $p \times 1$ — that is $X^t y$ is a p-vector. Similarly, $X^t X$ is the product of a $p \times n$ matrix and an $n \times p$ matrix so $X^t X$ is $p \times p$. These dimensions are written in under the objects in the normal equations. The normal equations form a system of p linear equations in p unknowns, so, as long as $X^t X$ is not singular, we can solve the system for a — usually we can.

The solution to the normal equations gives the best prediction of the response in the sense of SSE.

As an example, suppose we measure water and sunlight for each of 10 sunflowers we grow from seed. The amounts each flower receives are given in the *design* matrix, X, below, while the heights of the sunflowers are given in the response vector, y where

$$X = \begin{pmatrix} 1 & 2 \\ 1 & 4 \\ 2 & 3 \\ 0 & 1 \\ 1 & 0 \\ 5 & 2 \\ 5 & 1 \\ 4 & 1 \\ 6 & 5 \\ 1 & 2 \end{pmatrix} \qquad y = \begin{pmatrix} 9 \\ 18 \\ 14 \\ 2 \\ 0 \\ 17 \\ 12 \\ 17 \\ 29 \\ 5 \end{pmatrix}$$

After creating the X matrix and y vector in R we can solve this through

```
a = solve(t(X) %*% X , t(X) %*% y)
```

where a_1 is the resulting coefficient for water and a_2 is the resulting coefficient for sunlight.

As another example we return to the flower seeds case presented above. When we looked at that example before we learned that with 4 different mixes of 4 different kinds of flowers we could choose the appropriate quantity to buy of each kind of mix so that we end up with any desired quantity of the four different flower types. We did this by solving a 4x4 matrix equation.

Now suppose we only have the first three flower vendors available to us. If we tried to choose quantities to buy from the first three vendors so we get 3 ounces of poppies, 4 ounces of lupine, 5 ounces of daisies, and 6 ounces of

phlox, just as before, then we try to solve the matrix equation

$$\begin{pmatrix} .70 & .10 & .25 \\ .20 & .20 & .25 \\ .10 & .10 & .25 \\ .00 & .60 & .25 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 4 \\ 5 \\ 6 \end{pmatrix}$$

It isn't possible to solve this matrix equation — we have more equations (4) than unknowns (3). But we can try to do the best we can. That is, we can see how close to the target of (3,4,5,6) we can get in the sense of Euclidean distance. This is what the normal equations do: to find the closest approximation to our target we solve

$$\begin{pmatrix} .70 & .10 & .25 \\ .20 & .20 & .25 \\ .10 & .10 & .25 \\ .00 & .60 & .25 \end{pmatrix}^t \begin{pmatrix} .70 & .10 & .25 \\ .20 & .20 & .25 \\ .10 & .10 & .25 \\ .00 & .60 & .25 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} .70 & .10 & .25 \\ .20 & .20 & .25 \\ .10 & .10 & .25 \\ .00 & .60 & .25 \end{pmatrix}^t \begin{pmatrix} 3 \\ 4 \\ 5 \\ 6 \end{pmatrix}$$

which is easy to do on the computer, resulting in the solution

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -2.96 \\ 1.85 \\ 19.11 \end{pmatrix}$$

How close do we get to our target? We observe that

$$\begin{pmatrix} .70 & .10 & .25 \\ .20 & .20 & .25 \\ .10 & .10 & .25 \\ .00 & .60 & .25 \end{pmatrix} \begin{pmatrix} -2.96 \\ 1.85 \\ 19.11 \end{pmatrix} = \begin{pmatrix} 2.88 \\ 4.55 \\ 4.66 \\ 5.88 \end{pmatrix}$$

$$\approx \begin{pmatrix} 3 \\ 4 \\ 5 \\ 6 \end{pmatrix}$$

You can easily verify that any other candidate for the vector of unknowns produces quantities of flowers that are further from the target in the sense of Euclidean distance.

Geometric Interpretation of Normal Equations

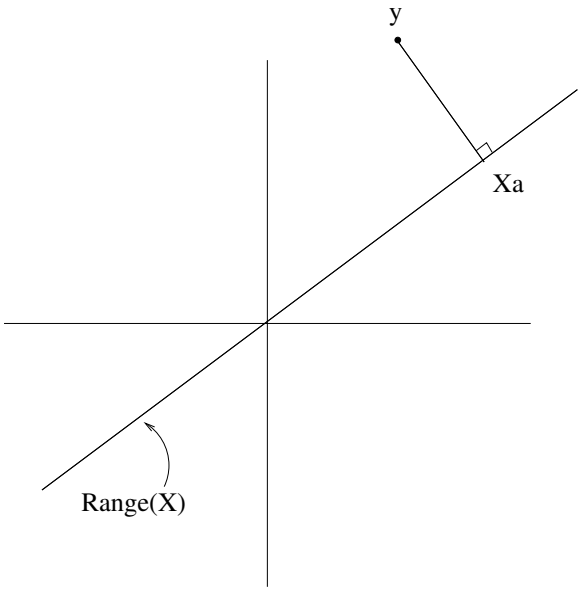
Before going ahead with the normal equations in regression, let's first consider them in in a geometric context so we can *visualize* what they do. We first consider the simplest possible situation where X is a 2×1 matrix and y is a 2-vector. Geometrically speaking, the normal equations identify the a so that Xa is as close to y as possible. First we consider the "world" of vectors Xa where a is anything — in other words, the *range* X. In this simple case a is a 1-vector — just a single number — since X has only 1 column. If

$$X = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

then the range of X is the collection vectors

$$\{a \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} : a \in \Re\}$$

(remember the interpretation of matrix multiplication as a linear combination of the columns of X). We recognize this as a line in the plane that goes through the origin. We have learned that the solution to the normal equations is the vector minimizing $||y - Xa||$ — the vector a so that Xa is as close to y as possible. This is indicated in the picture below. The closest point to y in the range of X must be the orthogonal projection of y onto the range of X.



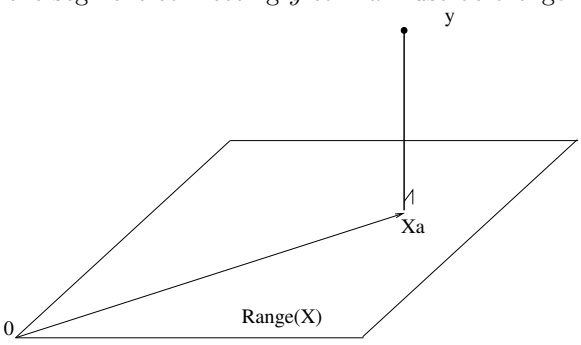
Suppose we consider the situation where X is a 3×2 matrix and y is a 3-tuple. If we write the columns of X as

$$X = \begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \end{pmatrix} = (X_{\cdot 1}, X_{\cdot 2})$$

then the range of X is

$$\{Xa = a_1 X_{\cdot 1} + a_2 X_{\cdot 2} : a_1, a_2 \in \Re\}$$

which gives a plane in 3-space. We know that the if a is the solution to the normal equations, $X^t y = X^t X a$, then Xa is as close to y as possible. Thus the segment connecting y to Xa must be orthgonal to the plane, as in the figure.



We now use R to illustrate what the normal equations do. In the following short program, normal_eq_local_opt.r, we choose an arbitrary 3×2 matrix, X, and an arbitrary 3-vector y. At this point we may believe that the solution to normal equations, that is, the vector, a, solving $X^t y = X^t X a$ minimizes $||y - Xa||$. That is Xa is closer to y than any other Xa'. To test this out, we look at a sampling of vectors a' near the solution to the normal equations and plot the squared distance to y for every Xa'. The plot shows that the minimum value occurs with a — the one in the middle of the plot.

```
X = matrix(c(1,3,9,5,4,2),byrow=F,ncol=2)   # any 3x2 matrix
y = c(5,2,1)     # any 3-dim vector
a = solve(t(X) %*% X, t(X) %*% y)  # solution to normal eqns.
n = 100
e = matrix(0,nrow=n,ncol=n)
lim = 5
grid1 = seq(-lim,lim,length=n);
```

```
grid2 = seq(-lim,lim,length=n);
for (i in 1:n) {
    for (j in 1:n) {
        aa = a + c(grid1[i],grid2[j])
        d = y - X %*% aa
        e[i,j] = sum(d*d)
    }
}
contour(e)
```

3.3.1 Simple Linear Regression Revisited

We now reformulate the simple linear regression problem in terms of what we have learned about linear algebra. Recall that in simple linear regression we are given data $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ and wish to fit the best line $a_1 x + a_2$ to the points. (Here we call the line parameters (a_1, a_2) rather than (α, β) as before). We seek the line parameters so that the sum of squared errors

$$SSE = E(a_1, a_2) = \sum_{i=1}^{n} (y_i - (a_1 x_i + a_2))^2$$

is minimzed.

Suppose we define the matrix X by

$$X = \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{pmatrix}$$

Then, write $a = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}$ we see that

$$Xa = \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} a_1 x_1 + a_2 \\ a_1 x_2 + a_2 \\ \vdots \\ a_1 x_n + a_2 \end{pmatrix}$$

Letting

$$y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

we get

$$y - Xa = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} - \begin{pmatrix} a_1 x_1 + a_2 \\ a_1 x_2 + a_2 \\ \vdots \\ a_1 x_n + a_n \end{pmatrix} = \begin{pmatrix} y_1 - (a_1 x_1 + a_2) \\ y_2 - (a_1 x_2 + a_2) \\ \vdots \\ y_n - (a_1 x_n + a_n) \end{pmatrix}$$

so

$$||y - Xa||^2 = \sum_{i=1}^{n} (y_i - (a_1 x_i + a_2))^2 = SSE$$

We have learned that the a that minimizes the SSE above is the solution to the normal equations. That is, the minimizing a solves $X^t y = X^t X a$. In summary, if we choose X in the right away (as we did above) then the problem of fitting a line to the data is just a case of the normal equations.

This is a much simpler way to fit the line to our data, and much easier to remember. If we wanted to do this in R for the cars data, as before, all we need to do is the following

```
data(cars)
x = cars$speed
y = cars$dist
n = length(x);
plot(x,y,xlab = "speed (mph)",ylab="stopping dist (feet)")

# now solving the normal equations ...

X = cbind(x,rep(1,n))    # cbind joins the column x with the column of 1s to get our X matrix
a = solve(t(X) %*% X , t(X) %*% y);
abline(a[2],a[1])
```

One should examine the result we get for a in this method and verify that the line parameters are identical to what we computed before.

3.3.2 Polynomial Regression and Seasonal Regression

So far we have used the normal equations to duplicate our initial result for simple linear regression. If this were all the normal equations were good for there wouldn't be much point of introducing all of the new machinery. But we will now see that the new approach is far more flexible than where we began.

Suppose we are given $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, as with the Cars data. Imagine that, rather than fitting a line to the data, we would like to fit a *quadratic*. That is, we would like to fit our data by $\hat{y} = a_0 + a_1 x + a_2 x^2$. Such quadratic curves give parabola shapes. Even though we are no longer talking about *lines* the linear algebra approach we have developed applies equally well to this situation. If we begin the the equation for the ith data point as

$$\hat{y}_i = a_0 + a_1 x_i + a_2 x_i^2$$

we can expand this out as a system of equations:

$$\begin{array}{ccccccc} \hat{y}_1 & = & a_0 1 & + & a_1 x_1 & + & a_2 x_1^2 \\ \hat{y}_2 & = & a_0 1 & + & a_1 x_2 & + & a_2 x_2^2 \\ \vdots & \vdots & \vdots & & \vdots & & \vdots \\ \hat{y}_n & = & a_0 1 & + & a_1 x_n & + & a_2 x_n^2 \end{array}$$

which can be written in matrix format as

$$\underbrace{\begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_n \end{pmatrix}}_{\hat{y}} = \underbrace{\begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{pmatrix}}_{X} \underbrace{\begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix}}_{a}$$

Consider the names given under the objects above, we have written the system of equations as

$$\hat{y} = Xa$$

thus representing the problem in a familar way. If y is the vector of observed responses, we can the vector a that minimizes the sum of squared errors:

$$SSE = \sum_{i=1}^{n} (y_i - (a_0 + a_1 x_i + a_2 x_i^2))^2 = ||y - Xa||^2$$

by solving the normal equations $X^t y = X^t X a$.

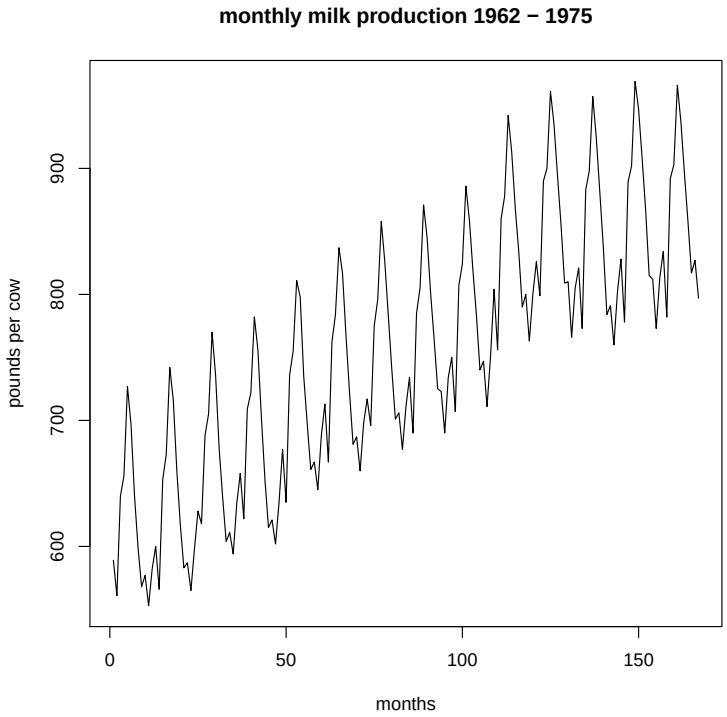
There is nothing at this idea that is limited to fitting quadratic curves. We could easily fit a 3rd order or higher polynomial to the data simply by adding more columns of X and more coefficients to a.

The R example polynomial_regression_synthetic.r shows a straightforward application of these ideas on synthetic data. In this case we just take our x's to be evenly spaced and choose the y's according to our quadratic model, though adding noise. The example shows that, whatever the choices of the coefficients, or even the order of the polynomial, the normal equations recover a good fit of the data and a reasonably accurate estimate of the true coefficients.

The R example polynomial_regression_lake_erie.r shows an example of polynomial regression on real data — the average levels of Lake Erie, over the period of years from 1921 to 1971. In this case we don't know what the *true* model is — is there such a thing here? But we can still fit the data using different orders of polynomial regression, as the example illustrates.

Seasonal Regression

Consider the data below showing milk production in a certain region over the years of 1962 to 1975. Over this time period there were monthly measurements. The total amount of milk production grew significantly during this time, but our data set focuses on the efficiency of the process, so we use the *per capita* measurement of "pounds per cow" — the total number of pounds of milk *divided* by the number of cows involved. There are two things that are immediately evident from the graph below. First of all, there is a seasonal pattern of production. This is as we would expect, since there is a cyclic annual pattern with many things we observe in the natural world. The second observation is that the overall level of efficiency, in pounds per cow, increased over this time period as the farming techniques improved. We wish to use linear regression to model these data.



Suppose our monthly pounds-per-cow data are given by $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$. In our polynomial regression section above, we saw we could use linear regression, in particular, the normal equations, to fit a *polynomial* to a data set. We do this by creating an X matrix with one column for each predictor variable. In the case of a polynomial the predictor variables are $1, x, x^2, \ldots, x^p$. That is,

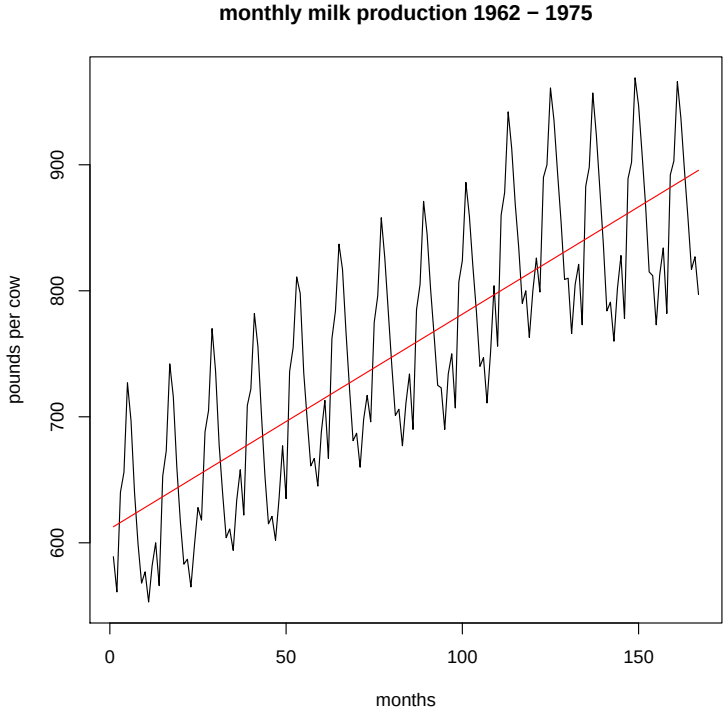
$$X = (1, x, x^2, \ldots, x^p) = \begin{pmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^p \\ 1 & x_2 & x_2^2 & \ldots & x_2^p \\ \vdots & \vdots & \vdots & \vdots & \\ 1 & x_n & x_n^2 & \ldots & x_n^p \end{pmatrix}$$

The essential idea is that we can create an X matrix where the columns are any functions of x we like, as is apppropriate for the case at hand. In the current case a polynomial doesn't seem like a reasonble choice, so we start simply and fix things up as we go along.

The first, and most straightforward thing we might try is to fit a line to the data. We do this in the usual way of constructing our X matrix with two columns, one for the x vector — the times at which the measurements were taken, and a column of 1's as we did in the previous section. Using the R fragment below, this leads to the following fit of the data:

```
cow = read.csv("data/monthly-milk-production-pounds-p.csv",stringsAsFactors=FALSE, sep=",");
n = 167   # number of months we use
y = cow[1:n,2]  # pounds per cow (response variable)a
x = 1:n
plot(x,y,type="l",main="monthly milk production 1962 - 1975",xlab="months",ylab="pounds per cow");
X = cbind(rep(1,n),x);   # simple linear regression (exactly as before)
a = solve(t(X) %*% X, t(X) %*% y);   # solve the normal equations
yhat = X %*% a;    # our predicted values
lines(x,yhat,col=2);
```

The result of this approach is shown in the plot below. There is a lot to be said in favor of this simple approach: Even though there is a complex seasonal aspect to the data, ignoring this pattern still leads to a fit that captures the overall trend of the data over time.

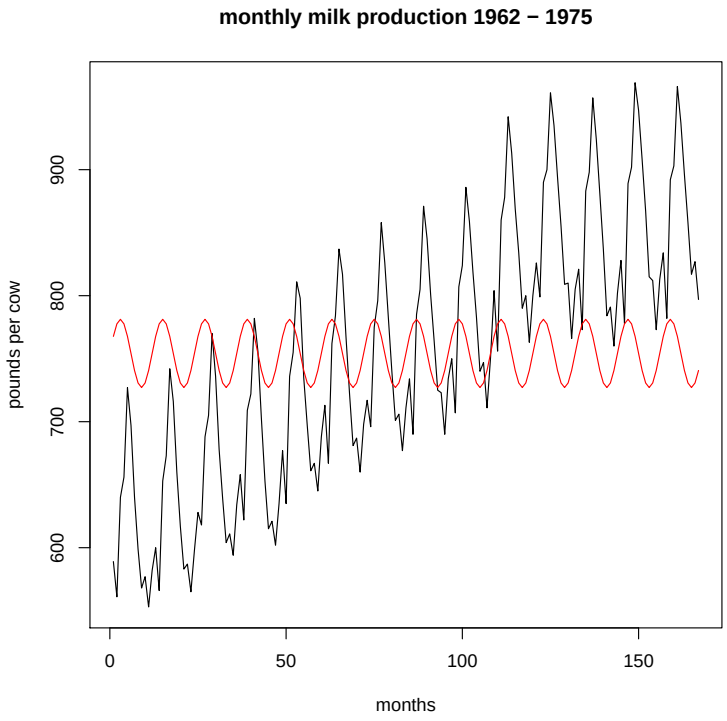


But what if we also wanted to model the seasonal variability? The seasonal variability is *periodic* — meaning that it is a cyclical pattern that repeats over and over. We know the *period* of this cycle must be one year, or, since our data consists of monthly measures, the period must be 12. What is a function we could add in that has period 12? The most obvious choice would be something like a "sine wave," $\sin(x)$. The problem here is that the sin has period 2π while we want our modeling function to have period 12. A simple rescaling of the sin does this — we will take $\sin(2\pi x/12)$ to be our new modeling function. Since clearly this is not enough to accurately capture the nature

of the data, we will also add in a constant term, leading to the X matrix

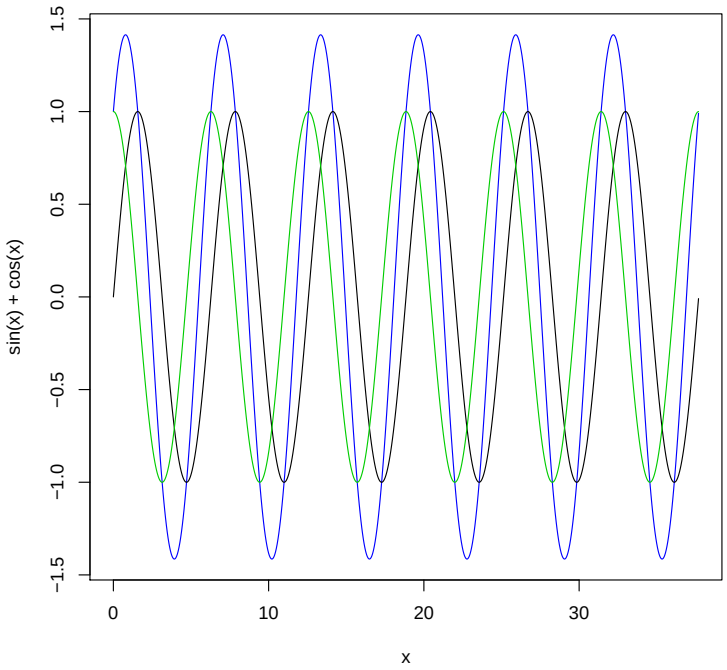
$$X = (1, \sin(2\pi x/12)) = \begin{pmatrix} 1 & \sin(2\pi x_1/12) \\ 1 & \sin(2\pi x_2/12) \\ \vdots & \vdots \\ 1 & \sin(2\pi x_n/12) \end{pmatrix}$$

Substituting this matrix into the code above — the generic regression code — leads to the model fit depicted below.



This obviously didn't work very well! Not only is the amplitude (height of oscillation) of the sine wave wrong, but the peaks don't occur in the right place. That is, the sin model has its peak occurring 1/4 of the way through its cycle, in March, since this is just what the sine does, while the milk production seems peak a couple of months later — we don't know exactly when. How would we construct a periodic function that can *shift* its peak to the right place?

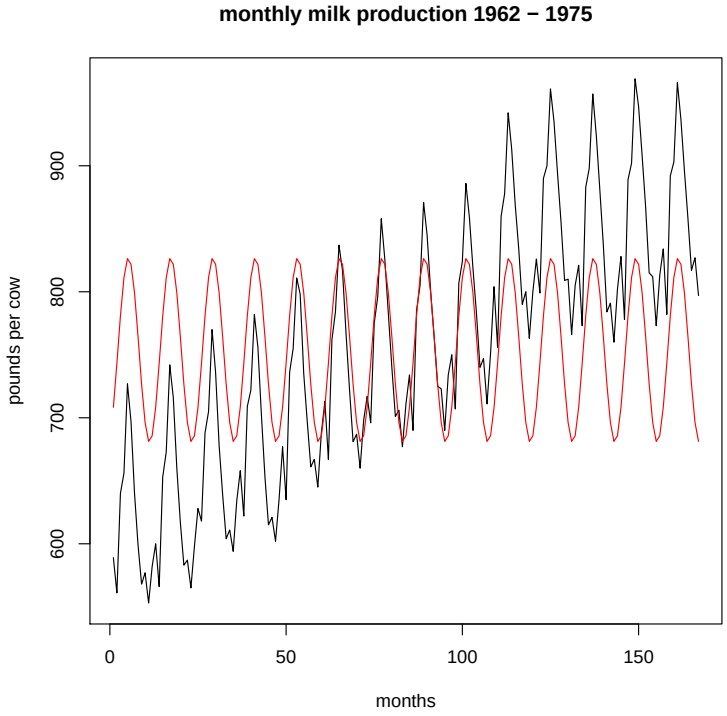
Here we introduce an interesting little bit of trigonometry that may not be familiar. We know what the graph of $\sin(x)$ looks like, and also what the graph of $\cos(x)$ looks like. These are both periodic functions with period 2π. Really the sin and cos look pretty much the same, except that each is shifted in relation to the other. But what does $\sin(x) + \cos(x)$ look like? These three graphs are shown in the figure below. We see that the sum of sin and cos has the same basic shape as both sin and cos but is *shifted* and *scaled*. While we won't go into the details of the math, the result is really more general than this. If we look at the function $f(x) = a_1 \sin(x) + a_2 \cos(x)$, where the constants a_1, a_2 can be *anything* — even negative — the result is just a shifted and scaled version of the sin (or cos). In fact, *any* shifted and scaled version of the sin can be created this way.



This tells us how to resolve our "shift" problem encountered before — we will create a model that can fit an arbitrary shift in the peak by including both sin *and* cos as predictors. That is, our prediction matrix now becomes

$$X = (1, \sin(2\pi x/12), \cos(2\pi x/12)) = \begin{pmatrix} 1 & \sin(2\pi x_1/12) & \cos(2\pi x_1/12) \\ 1 & \sin(2\pi x_2/12) & \cos(2\pi x_1/12) \\ \vdots & \vdots & \vdots \\ 1 & \sin(2\pi x_n/12) & \cos(2\pi x_n/12) \end{pmatrix}$$

The fit of this new model is shown in the figure below.



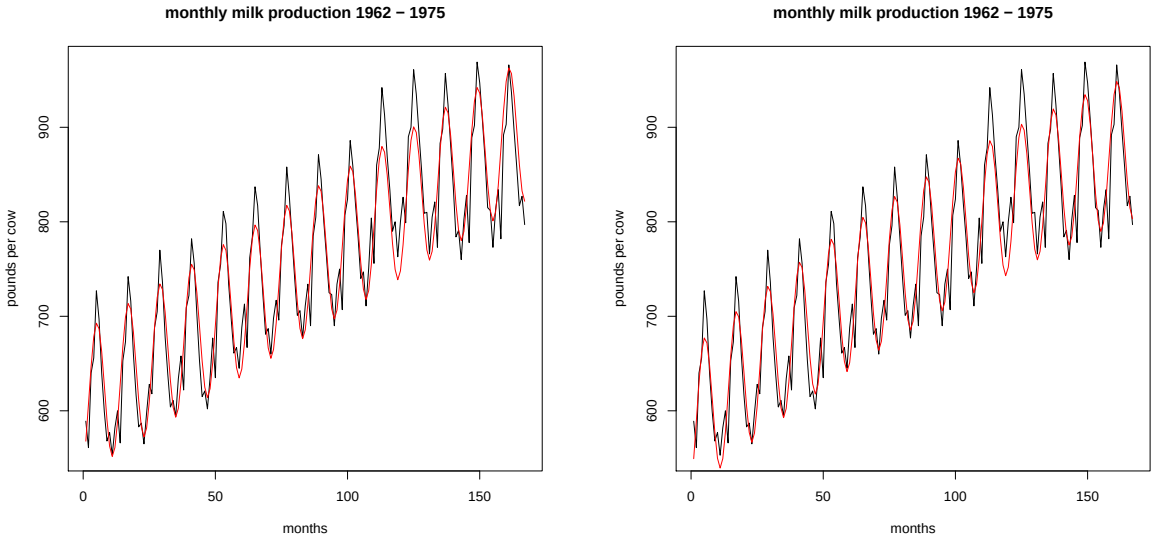
Now we have captured the appropriate shift and amplitude of the seasonal variation in milk production. We can add in the trend by including a linear term with

$$X = (1, x, \sin(2\pi x/12), \cos(2\pi x/12)) = \begin{pmatrix} 1 & x_1 & \sin(2\pi x_1/12) & \cos(2\pi x_1/12) \\ 1 & x_2 & \sin(2\pi x_2/12) & \cos(2\pi x_1/12) \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & \sin(2\pi x_n/12) & \cos(2\pi x_n/12) \end{pmatrix}$$

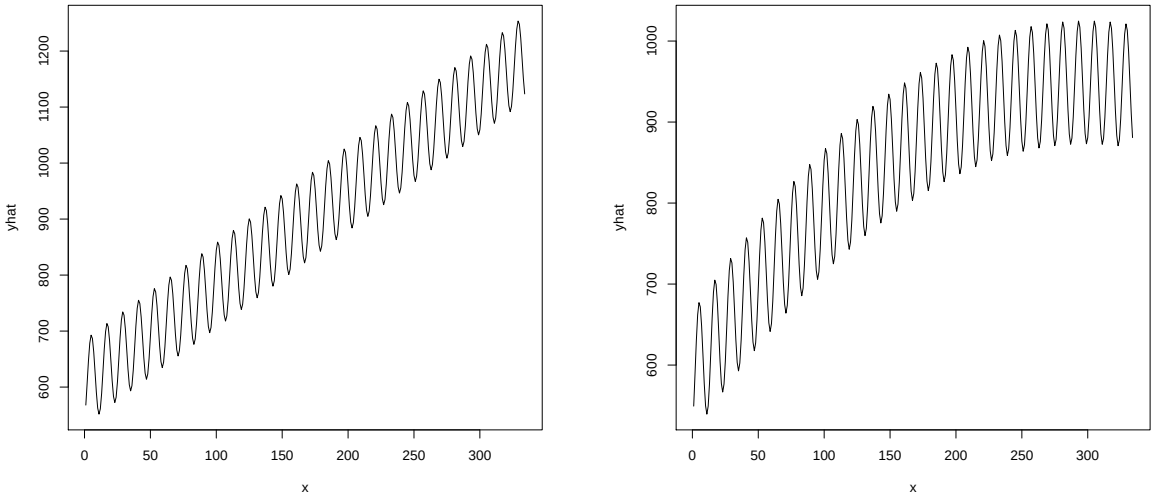
or even one with a quadratic fit:

$$X = (1, x, x^2, \sin(2\pi x/12), \cos(2\pi x/12)) = \begin{pmatrix} 1 & x_1 & x_1^2 & \sin(2\pi x_1/12) & \cos(2\pi x_1/12) \\ 1 & x_2 & x_2^2 & \sin(2\pi x_2/12) & \cos(2\pi x_1/12) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \sin(2\pi x_n/12) & \cos(2\pi x_n/12) \end{pmatrix}$$

resulting in the figures below:



To quote Yogi Berra, the former catcher and manager of the New York Yankees, "Predicton is hard, especially about the future." This is certainly true though our models allow us to try. Using our fitted parameters and just extending our range of values of x by a factor of two allows us to predict what might happen inx the next 13 years of milk production. These predictions are given in the figures below, using both the linear and quadratic models discussed. What do you think about the validity of these predictions?



3.3.3 Regression with Multiple Predictor Variables

In the previous section we examined regression problems where our data are $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ where both the response y_i and the predictor x_i are single numbers. In this case we saw that we could *construct* other predictor variables from x_i as *functions* of x_i. In the current section we will look at the situation in which we have multiple predictors for each response x_i. In this case our data matrix takes the generic form

$$X = \begin{pmatrix} x_{11} & x_{12} & \ldots & x_{1p} \\ x_{21} & x_{22} & \ldots & x_{2p} \\ \vdots & \vdots & \ldots & \vdots \\ x_{n1} & x_{n2} & \ldots & x_{np} \end{pmatrix}$$

Thus we view the collection of all the p predictor variables for the n instances as an $n \times p$ matrix. Each row if the data matrix X is an instance, while each column is a feature or predictor variable. We can write the vector of responses, one for each of the n instances, as a vector y where

$$y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

For example, the predictor variables might be various things we measure about a plant, such as the amount of sunlight it gets, the amount of water, the amount of nitrogen, etc., while the response may be the height the plant grows to.

While the current situation regards the predictor variables a little differently than in the previous section, the main ideas are identical. We are interested in choosing regression coefficients $a_1, \ldots, a_p$ so that our estimate of the ith response, $\hat{y}_i$ is the linear combination

$$\hat{y}_i = a_1 x_{i1} + a_2 x_{i2} + \ldots + a_p x_{ip}$$

or, written more expansively

$$\begin{array}{ccccccccc} \hat{y}_1 & = & a_1 x_{11} & + & a_2 x_{12} & + & \ldots & + & a_p x_{1p} \\ \hat{y}_2 & = & a_1 x_{21} & + & a_2 x_{22} & + & \ldots & + & a_p x_{2p} \\ \vdots & \vdots & \vdots & & \vdots & & \vdots & & \\ \hat{y}_n & = & a_1 x_{n1} & + & a_2 x_{n2} & + & \ldots & + & a_p x_{np} \end{array}$$

This can easily be written in matrix form by letting

$$\hat{y} = \begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_n \end{pmatrix}$$

$$a = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{pmatrix}$$

and writing

$$\hat{y} = Xa$$

We would like to get our predictions $\hat{y} = Xa$ as close to y as possible in the sense of SSE (which is the same as Euclidean distance. We know we can do this by solving the normal equations

$$X^t y = X^t X a$$

for a, thus identifying our regression coefficients.

Before looking at real data it may be helpful to see this in action when we *know* the xmodel the data come from. This is demonstrated in several_var_synth_regression.r

Chatterjee and Price Attitude Data

This next example is from a popular book: *Regression Analysis by Example* by Chatterjee and Price. The example treats a large financial organization with 30 different departments. The company was interested in improving their employees' satisfaction with the working environment, hoping to retain their best employees. To do this the company wanted to understand which are the important factors that contribute to their employees' satisfaction, thus the undertook the following regression analysis.

35 employees were surveyed from each of the departments and asked 6 yes/no questions, as well as one final yes/no question on overall satisfaction. The 6 questions asked if the employee was satisfied with the handling of

1. Employee complaints
2. Special privileges
3. Learning opportunites
4. Performance-based raises
5. Degree of criticism
6. Possibilities for advancement

For each of the 6 questions, as well as the final overall satisfaction question, the 35 answers were pooled to get a *satisfaction percentage*. Thus, for all questions, high numbers indicate satifaction while low numbers indicate dissatisfaction.

We seek regression coefficients, $a_1, \ldots a_6$ so that, for each department, the overall satisfication can be approximated as a linear combination of the 6 satisfaction scores. That is,

$$\begin{array}{c} \text{dept. 1} \\ \text{dept. 2} \\ \vdots \\ \text{dept. n} \end{array} \begin{pmatrix} \text{overall satisfaction} \\ \text{overall satisfaction} \\ \vdots \\ \text{overall satisfaction} \end{pmatrix} \approx \begin{pmatrix} \text{complaints} & \ldots & \text{advancement} \\ \text{complaints} & \ldots & \text{advancement} \\ \vdots & \vdots & \vdots \\ \text{complaints} & \ldots & \text{advancement} \end{pmatrix} \begin{pmatrix} a_1 \\ \vdots \\ a_6 \end{pmatrix}$$

The actual analysis of these data is taken up in the R example attitude_regression.r.

Class Example: Look at the state.x77 R dataset, predicting murder rate from frost, graduation rate, and illiteracy. Should we include the constant term (columns of 1's?) Which single variable predicts murder rate best?

3.4 Regression and Overfitting

We have seen that some classifiers are prone to *overfitting* — meaning that the classifier is overly adapted to the training data, yielding good performance on the training data, but generalizing poorly when we look at different data. Classifiers with many free parameters, such as tree classifiers, are particulary prone to this problem. Now we are studying regression rather than classification, though the same potential for overfitting our model happens here too.

$n \times n$ Systems are Solvable

To give some intuition here, first consider the generic matrix equation

$$Xa = y$$

where X is an $n \times n$ matrix, and a, y are n-vectors. Recall our earlier discussion of the $n = 2$ case. In this case the two equations of the matrix equation

$$\begin{aligned} x_{11}a_1 + x_{12}a_2 &= y_1 \\ x_{21}a_1 + x_{22}a_2 &= y_2 \end{aligned}$$

each describe a collection of points (a_1, a_2) that forms a line in the plane. Since two lines usually intersect in a point, the matrix equation has a unique solution. By "usually", we mean when the matrix is not singular.

The same holds in 3-space. In this case X is a 3×3 matrix, while a and y are 3-vectors. Now the three equations of the matrix equation each describe a *plane*. Since the intersection of three planes is a point (when X is non-singular) we again have a unique solution for a. In general, as long as X is non-singular there is a unique solution to the n-dimensional matrix equation. The R example square_system_of_linear_equations.r gives some computational validation of this assertion. In this example we create random matrices and vectors, still finding that we always succeed in indentifying a solution to the system.

Overfitting Example

We now consider these ideas in the context of linear regression. Consider a regression scenario similar to the classification problem we examined in tree_overfitting.r where the class label was *independent* of the predictor variables, thus there was no chance of any meaningful prediction. Here again the response variable y will be independent of the predictors. The experiment is simple enough that we list the R code below:

```
n = 100
Z = matrix(rnorm(n*n),nrow=n,ncol=n);
y = rnorm(n)  # there is no relationship between cols of Z (predictors) and y (response)
for (i in 1:n) {          # i will be the number of variables used
    X = Z[,1:i]           # X uses the first i predictors
    a = solve(t(X) %*% X, t(X) %*% y);
    yhat = X %*% a
    error = y-yhat
    sse = sum(error*error)
    cat(i, " predictor variables, sse = ", sse, "\n");
}
```

Here we consider $p = 100$ randomly generated predictor variables (columns) in the matrix Z, with a randomly generated response y. $n = 100$ too, so Z has 100 rows while y also has 100 elements. Without even doing the experiment we know that, whatever happens with our estimate of regression coefficients, a, we should expect terrible performance when we look at new data from this same model — the response is independent of the predictors so the situation is hopeless.

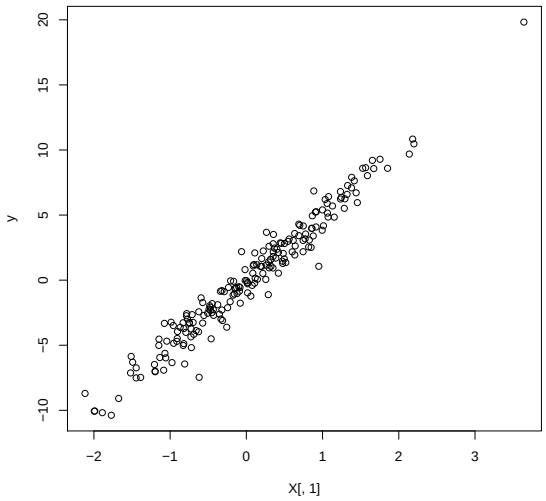
In the example we consider larger and larger collections of possible predictor variables (columns of Z). In the first iteration through the loop we try to predict y using only the first column of Z. In the second iteration we use the first *two* columns of Z. We continue with this pattern, using the first i columns in the ith iteration. In each iteration we measure the degree to which we fail to predict y. In particular, we use the sum of squared errors (squared Euclidean distance) as our meausure. The example shows that, as we add more and more predictor variables, the sum of squared errors is driven steadily to 0. In the final iteration, using all 100 predictor variables, we have the numerical equivalent of *no* error in predicting y. Even though we knew the problem was hopeless we seem to have constructed a perfect predictor. Of course, our prediction will not work on different data as we saw in the analogous classification example. You should run the code above to verify that the prediction error (SSE) is driven all the way to 0.

A moment's reflection reveals that this had to happen. When we use all predictor variables, predicting y amounts to solving the 100×100 system of linear equations $y = Za$. Since we know this is possible, we know we can get perfect prediction (on the training data!).

The above example shows how linear regression, like most model fitting methods, has the potential to overfit. To pursue this further, we want to consider a situation, unlike the preceding, where there *is* a right answer. In the following, illustrated in the program one_useful_variable.r case our response depends on *only one* of the p predictor variables. The other variables are *independent* of the response, much like the extra variables in the digit recognition example we addressed using tree classifiers. Consider the R code below, which creates the synthetic data:

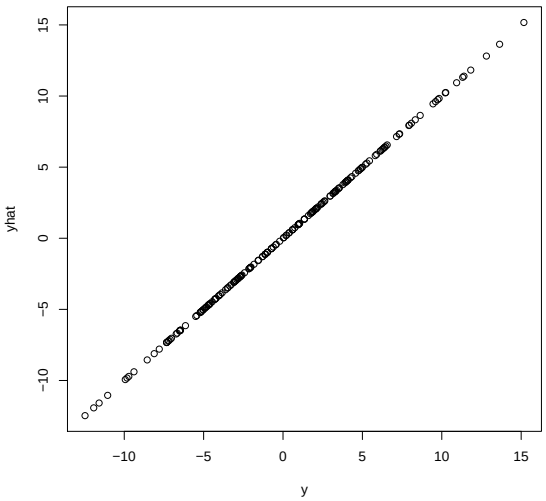
```
n = 200
p = 200                         # p = n means likely overfitting
X = matrix(rnorm(n*p),nrow = n, ncol=p);
y = 5*X[,1] + rnorm(n);         # only 1st variable useful for predicting y
plot(X[,1],y);                  # can see obvious relation between first predictor and y
```

The plot is shown below the clearly demonstrates the connection between the first variable and the response.



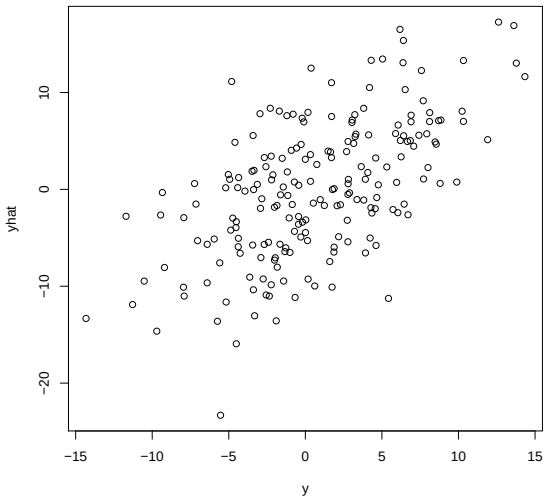
We can see from the R code above that we have as many predictor variables as we have instances, ($p = n = 200$) so the situation will be prone to overfitting if we do nothing to stop this. Proceeding directly by solving the normal equations gives great prediction on the training data, as the following plot demonstrates:

```
a = solve(t(X) %*% X, t(X) %*% y);
yhat = X %*% a;
plot(y,yhat);    # perfect prediction.  should we be happy ..... ?
```



As usual, the problem surfaces when we look at new data, identical to our original data in the way it is created. The code below shows such data and the result of using our prediction coefficients on this new data. We can see from the plot that the results are not good. We overfit the data.

```
X = matrix(rnorm(n*p),nrow = n, ncol=p);  # try new data from same model
y = 5*X[,1] + rnorm(n);
yhat = X %*% a;                 # N. B. we use same a as above.
plot(y,yhat);                   # prediction not so good
```



It is worth considering for a moment what we should *want* to happen in this cases. Here we have only one variable that is of any use in predicting the response, so we hope that the coefficients of all the other variables will be 0. Failing that, we hope they would at least be small compared to the coefficient of the first variable to avoid overfitting.

Ridge Regression

In an attempt to avoid the overfitting we pursue a strategy much like what we did with trees. Our current regression strategy tries only to optimize the *fit* to the data, as measured by sum of squared errors. That is, we have learned that our regression coefficients, a, obtained as the solution to the normal equations minimize SSE. We have seen, both in the case of trees, and now again with regression, that strategies that care only about how well the data are fit are prone to overfitting. So, as we did with trees, we introduced a *penalized* criterion to optimize. But what should this penalized criterion be?

Consider the regression equation

$$\hat{y}_i = a_1 x_{i1} + a_2 x_{i2} + \ldots + a_p x_{ip}$$

If some of the a coefficients are 0, the associated predictor variables are not used in the prediction. It might make sense to use the number of non-zero regression coefficients as our complexity penality, thus paying a fixed cost for each predictor variable we actually use. This is analogous to the split penalty we introduced with classification trees. While this makes some intuitive sense, it is hard to compute the resulting penalized fit. Instead we will take a penalty $\lambda > 0$ and use $\lambda||a||^2$ as our penalty term. Thus we "bias" or "drive" our regression coefficients toward the more "neutral" 0 values.

This technique is known as Ridge Regression. Ridge regression chooses the regression coefficients by minimizing the penalized criterion, $H(a)$, defined by

$$H(a) = \underbrace{||y - Xa||^2}_{\text{data fit}} + \underbrace{\lambda||a||^2}_{\text{complexity penalty}}$$

for some $\lambda > 0$. Like with trees, this criterion trades off a goodness of fit penalty, $||y - Xa||^2$, and a complexity penalty, $\lambda||a||^2$. That is, minimizing the goodness of fit terms encourages us to fit the training data as best we can, while minimizing the penalty term drives our regression coefficients toward 0 where the penalty term is least. The result is a tug-of-war between these two objectives.

One reason for the popularity of ridge regression is the ease with which we can solve the optimization above. One can show that the optimizing value of our penalized criterion solves the equation

$$(X^t X + \lambda I)a = X^t y$$

where I is the so-called "identity" matrix — the $p \times p$ matrix with 1's on the diagonal and 0's elsewhere. We can construct the solution to the ridge regression equations by solving the above system of equations, just as we do with the normal equations.

Continuing with the R example with only a single useful predictor variable we examine the results for different choices of the penalty λ.

```
lambda = .01   # .01 .1, 1,  10, 100 ...
a = solve(t(X) %*% X + lambda*diag(p), t(X) %*% y);   # ridge regression
yhat = X %*% a;
X = matrix(rnorm(n*p),nrow = n, ncol=p);  # try new data from same model
y = 5*X[,1] + rnorm(n);
yhat = X %*% a;
plot(y,yhat);
```

As we saw with trees, there is a characteristic pattern for the complexity penalty λ. When λ is large we are essentially not allowed to try to fit the data, so the resulting performance on new data is poor. As we decrease λ we our model fits the training data more and more, and we observe increasingly better performance. But, eventually, increasing λ enough results in overfitting, so the performance starts to get worse. We seek the "sweet spot" where we get the best generalization error.

How to find this sweet spot? The obvious thing is to do what we did with trees. We try out different values of the complexity parameter, seeking the value that gives the best generalization performance. We will not go into these details here, but the idea of the strategy should seem familiar.

3.4.1 Variable Selection

Ridge regression is one way to address overfitting with regression. In essense the idea is to "bias" or "pull" each regression parameter to the "neutral" or "unused" setting of 0. Variable selection is an entirely different way to deal with overfitting. In variable selection we try to identify a small subset of variables that perform well on the training data. The idea here is that if our subset of variables is small, we are finding a less complex solution, thus avoiding overfitting the data — the smaller the subset of variables, the less ability we have to fit the data at all, so there must be less ability to overfit.

We will discuss two highly similar variable selection methods known as *forward selection* and *backward selection*. Both are "greedy" algorithms, meaning that they make decisions that look good when the decision is made, but never revisit the decisions later on as the algorithm evolves. You will encounter many greedy algorithms in your study of optimization, because they are usually much simpler to implement than "global optimizaton" strategies — strategies that seek the best choice over all possibilities.

Forward Selection

The forward selection is a simple iterative algorithm that chooses variables incrementally. At each stage we choose the variable that gives the greatest decrease in SSE given the variables that have already been selected. In this discussion we assume that Z is an $n \times p$ matrix whose columns are the complete familily of possible predictor variables, and that y is the vector of responses. The algorithm is made more explicit in the following:

1. Find the single variable (column of Z) that "best explains" the response y. That is, find the single column of Z, so that when we let X be the single column, solve $X^tXa = X^ty$, and let $\hat{y} = Xa$ then

$$SSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

   is as small as possible.

2. Keeping the variable we just identified, form X by adding each remaining possible variable from Z. There are $p-1$ choices to consider here. For each possible 2-column X, choose the new variable that minimzizes the SSE.

3. Now choose as the 3rd variable the one that, when added to the first two, gives the smallest SSE in predicting y.

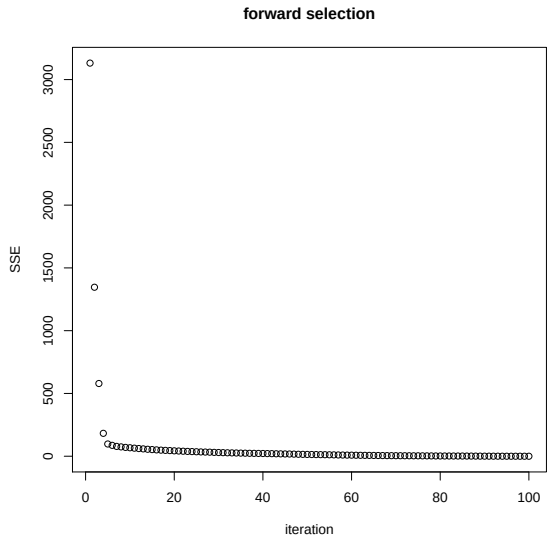
4. Continue this process until the change in SSE is no longer big enough to justify including the variable.

This procedure is formalized in the R code below, also in the program forward_selection.r. In this example we begin with a collection of 100 variables and 100 observations. That is Z is 100×100. We choose random values for the predictors, but construct y so that it only depends on the first 5 variables. More explicitly, our example models y as

$$y_i = 1x_{i1} + 2x_{i2} + 3x_{i3} + 4x_{i4} + 5x_{i5} + \epsilon_i$$

where ϵ_i is 0-mean and random. The last 95 variables are not used (though our algorithm doesn't know this).

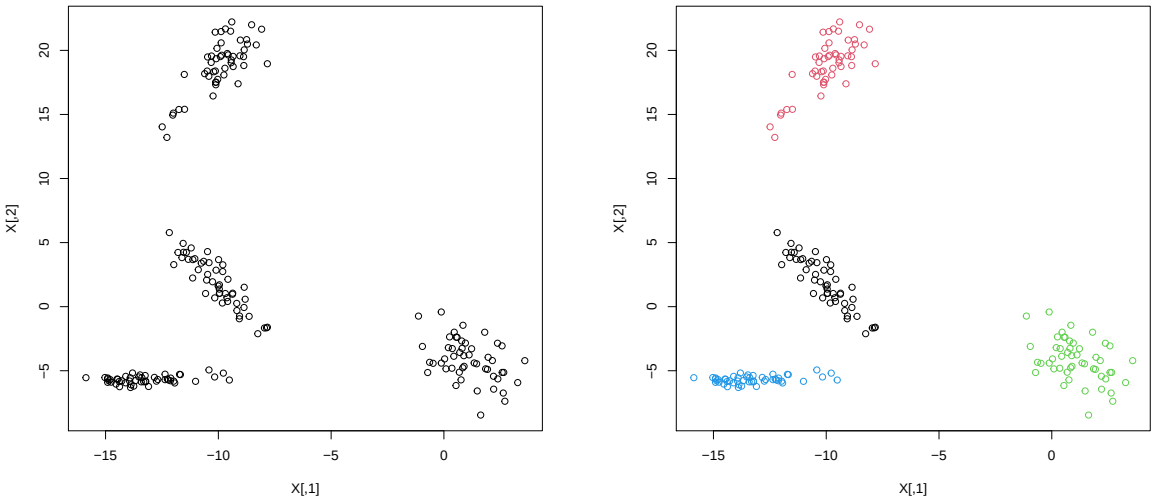
```
n = 100
p = 100;
Z = matrix(rnorm(n*p),nrow = n, ncol=p);

truea = c(1:5,rep(0,p-5))     # note that only the first 5 vars are useful in predicting y
y = Z %*% truea + rnorm(n);

used = rep(FALSE,p); # initially all varaibles available for selection (used[i] = FALSE)
var = rep(0,p); # var[j] will be variable chosen in jth round
bestsse = rep(10000000,p);      # bestsse[j] will be best sse from jth round
for (j in 1:p)  { # choose 1 variable each time through this loop
    for (i in which(used == FALSE)) {  # for all unused variables
       used[i] = TRUE;                 # temporarily mark as used
       X = Z[,used]; # take the "used" columns = used variables
       a = solve(t(X) %*% X , t(X) %*% y);
       yhat = X %*% a;
       error = y-yhat;
       sse = sum(error*error);
       if (sse < bestsse[j]) { # if we find a better sse, take it
          bestsse[j] = sse;

          var[j] = i;
       }
       used[i] = FALSE;        # now unmark as used
   }
   used[var[j]] = TRUE;  # claim the best variable for future iterations of loop
}
plot(bestsse)
```



In the figure above we can see that the SSE drops significantly as we add in the first 5 variables (which happen to be the variables 5,4,3,2,1) as one would expect given the known (to us) regression coeffecients. After that, there is still decrease in SSE as we add in the available useless variables, but the SSE decrease is small. This suggests that we might have a threshold and run the forward algorithm until the change in SSE is below the threshold.

Chapter 4

Clustering



Consider the left figure, above, in which we are given a collection of 2-dimensional feature vectors, plotting these in hopes of discovering their structure. From the figure we can see that there are four distict *groups* or *clusters* that divide the feature vectors. The figure on the right shows these exact same points, now *labeled* with four different clusters, indicated with four different colors. This is the essence of the *clustering* problem — we try to automatically divide a collection of feature vectors into groups so that all the members of a group are similar to one another.

There is a natural connection between the clustering problem and the classification problem we have already studied. In classification we try to estimate the class of a new feature vector given a training set composed of feature vectors with class labels assigned to each feature vector. In clustering we try to estimate class labels *without* the benefit of a labeled training set. For this reason classification is often refered to as *supervised learning*, whereas clustering is deemed *unsupervised learning*.

There are many different possible reasons one may want to cluster a collection of feature vectors. One is simply as a part of our *exploratory data analysis* — the phase where one first starts to grapple with the nature of a data set seeking to understand its structure. In this case it may be helpful to know if the feature vectors naturally divide into some number of classes. This suggests that our population is composed of several different groups, while this observation may be illuminating in and of itself, but also may suggestion modeling approaches that make use of this structure.

Clustering also arises in a process known as *vector quantization* (VQ). In vector quantization one seeks to simplify a collection of feature vectors, which may be high-dimensional, into clusters. After clustering, when future data vectors arise we simply report the clusters they belong to, rather than the complete feature vector. In this way we have reduced the high-dimension data points into numbers in $\{1, 2, \ldots, K\}$ where K is the number of clusters. Thus we have greatly simplified our dataset, making whatever process follows an easier one. For instance, consider the

previously discussed difficulities of creating a Bayes classifier with the original high-dimensional feature vectors vs. with the clustered data.

Finally clustering sometimes arises through trying to "fit" a model to data. We may find that, after clustering the data we have groups of data points that are easily modeled, while the collective data set is awkward to model. We will see an example of this kind, known as a Gaussian mixture model, later in this chapter.

4.1 The K-Means Algorithm

The most popular algorithm for clustering is known as the *K-Means* algorithm. It is a very simple algorithm that is easy to implement, "discovered" independently by many people. In the algorithm we wish to cluster a collection of n p-dimensional features, $x_1, \ldots, x_n$. The goal of the algorithm is to find which cluster each feature vector, x_i belongs to. We will write $k(i) \in \{1, 2, \ldots, K\}$ for the cluster x_i is assigned to.

The algorithm proceeds as follows:

K-Means Clustering Algorithm

1. Choose K and randomly choose K p-dimensional "prototypes", $m_1, m_2, \ldots, m_K$.

2. Assign x_i to the closest prototype. That is, let

$$k(i) = \arg\min_k ||x_i - m_k||^2$$

3. Recompute m_k as the mean of all vectors, x_i, assigned to cluster k. That is set

$$m_k = \frac{\sum_{i:k(i)=k} x_i}{N_k}$$

   where N_k is the number of feature vectors assigned to cluster k: $N_k = |\{i : k(i) = k\}|$.

4. Repeat steps 2 and 3 until there is no change in the cluster assignment $k(i)$.

An implementation of the K-means algorithm is given by the R program, kmeans_clustering.r. In addition to performing the algorithm, the R program provides visual annotation to track the evolution of the prototypes $m_1, \ldots, m_K$ and the clustering assignment $k(i)$. In particular, every time step 2 is performed we depict $k(i)$ by drawing each point x_i with the color of its cluster assignment. In this way, all vectors, x_i, assigned to class k will be drawn with the same color. In addition, after step 3 is performed we draw the locations of the cluster prototypes $m_1, \ldots, m_K$ denoting them with an x, also using color to distinguish them — the red x is the prototype of the red cluster, etc. To allow us time to inspect, the algorithm pauses after each iteration of step 2 and 3 until the "enter" key is pressed.

As we run the R program on a variety of randomly generated data sets we see that it frequently recovers the obvious clustering we seek — but not always.

4.1.1 K-Means as Optimzation Algorithm

Our initial presentation of the K-Means algorithm didn't state clearly what the algorithm tries to do. We saw that it often generates a reasonable clustering, though could there be a *better* one? This question is hard to answer without having a clear notion of what we mean by "better." In this section we will define an objective function that measures the "goodness" of a particular choice of clustering. We will show that K-Means is an algorithm that tries to optimize this objective function.

The Sample Mean Minimizes Sum of Squared Error

Before we go into the details of K-Means, we will introduce an important fact that is essential to understanding K-Means as an optimization algorithm. Suppose we have a collection of numbers $x_1, \ldots, x_n$ and consider the sum of squared differences from some other number, m:

$$SSE = \sum_{i=1}^{n} (x_i - m)^2$$

We are interested in finding the m that *minimizes* the SSE. If we define the *sample mean* to be

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

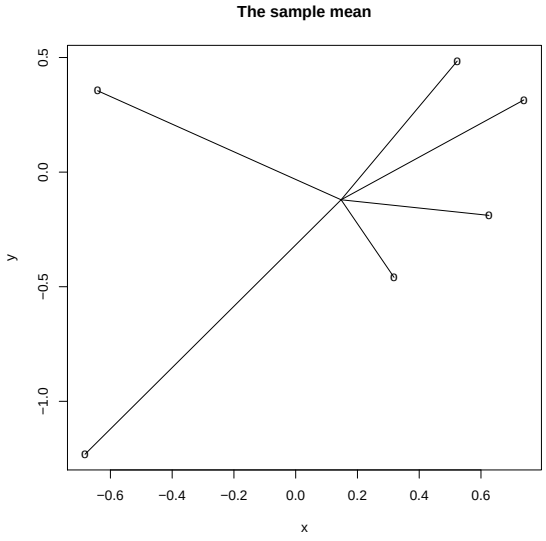
it is well known that taking $m = \bar{x}$ minimizes the SSE. That is

$$\bar{x} = \arg\min_m \sum_{i=1}^{n} (x_i - m)^2$$

In fact, this is easy to show. We write

$$\begin{aligned} SSE &= \sum_{i=1}^{n} (x_i - m)^2 \\ &= \sum_{i=1}^{n} (x_i - \bar{x} + \bar{x} - m)^2 \\ &= \sum_{i=1}^{n} (x_i - \bar{x})^2 + \sum_{i=1}^{n} (\bar{x} - m)^2 + 2 \sum_{i=1}^{n} (x_i - \bar{x})(\bar{x} - m) \\ &= \sum_{i=1}^{n} (x_i - \bar{x})^2 + \sum_{i=1}^{n} (\bar{x} - m)^2 \end{aligned}$$

where the last term disappers because $\sum_{i=1}^{n} (x_i - \bar{x}) = 0$. Since we are minimizing over m there is nothing we can do about the first term (m doesn't even appear there), while the 2nd term, hence the SSE, is minimized when $m = \bar{x}$. This is what we wanted to show.



It is worth remarking that this property also holds for *vectors*. That is, if $x_1, \ldots, x_n$ are p-dimensional vectors then,

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

still makes sense since it is defined in terms of addition and scaling of vectors – both operations we are familiar with from linear algebra. In the vector case we still have

$$\bar{x} = \arg\min_{m} \sum_{i=1}^{n} ||x_i - m||^2$$

where we are minimizing over the vector m. The argument for showing this is nearly identical to the one-dimensional case we showed, though we won't repeat the dervation for the vector case.

The figure shows an example of the sample mean for vectors where we draw a segment connecting each point to the sample mean. Another interesting fact about the sample mean is that the figure would "balance" on the sample mean if we constructed the geometrc object with identical weights at the points and weightless connectors.

4.1.2 The Objective Function

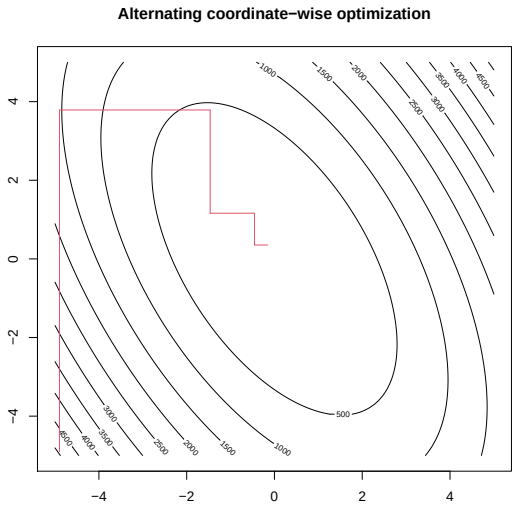
We said before that K-Means can be viewed as an optimization of an objective function, but what is that objective function? It is really a lot like the sum of squared errors we saw from linear regression. Suppose we have a cluster assignment $k(i)$, saying which cluster x_i is assigned to, as well as the cluster prototyptes, $m_1, \ldots, m_K$. We now consider the objective function

$$H = \sum_{k=1}^{K} \sum_{i:k(i)=k} ||x_i - m_k||^2 = \sum_{i=1}^{n} ||x_i - m_{k(i)}||^2$$

This makes sense as an objective function for clustering since we want points in the clusters to be close to one another, hence close to their cluster prototypes, $\{m_k\}$. This is exaclty what H measures — the sum of squared distances between each point x_i and its associated prototype m_k.

Recall that the K-Means algorithm had two steps to the iteration:

1. Assign each point to its closest prototype
2. Set the prototype for each cluster equal to the sample mean of that cluster



One can view the K-Means algorithms as an example of "coordinate-wise" optimzation. This is a technique for numerically optimizing a function of several variables by iteratively optimizing over each variable while holding the

others fixed. In this way we cycle through the variables over and over, solving simpler one-dimensional optimization problems. A pictorial example of this technique can be seen in figure above where we numerically optimize a function over two variables in a coordinate-wise fashion. The figure shows a sequence of points that would be "visited" by the early steps of the coordinate-wise optimzation.

We consider first step 1, which chooses the clustering function $k(i)$. Considering the "single sum" view of H in the equation above we see that for each term in the sum, indexed by i, assigning x_i to the closest prototype (what K-means does) chooses $k(i)$ to minimize $||x_i - m_{k(i)}||^2$. That is, choosing the cluster assignment as K-Means does minimizes H over all possible cluster assignments, while holding the $\{m_k\}$ fixed.

Now considering the 2nd operation, we view H in terms of the "double sum" above. We learned that $\sum_{i:k(i)=k}(x_i - m_k)^2$ is minimized when we take m_k to be the sample mean for the kth cluster. Thus the 2nd step, which sets m_k to the sample mean of the kth cluster, decreases each inner sum in H as much as possible. Thus we are choosing the $\{m_k\}$ to minimize H while holding the cluster assignment fixed.

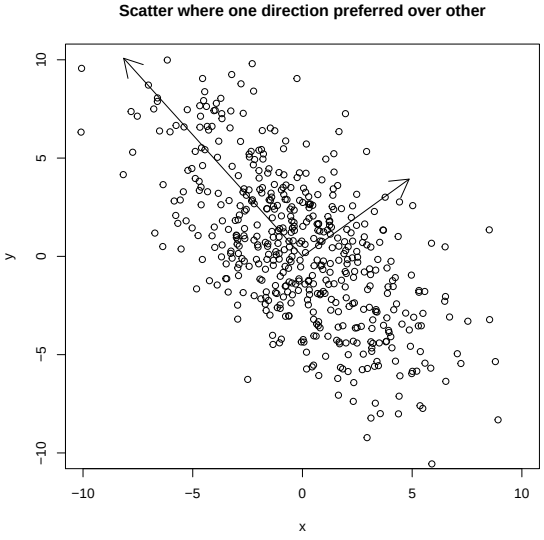
In summary, each step of the algorithm decreases H as long there is any change in the cluter assignment, $k(i)$, or the cluster prototypes, $\{m_k\}$. The algorithm terminates when there is no change in step 1, which means there can also be no change in step 2. Through this analysis we see that the algorithm cannot cycle through the same candidate cluster twice, since H decreases in each iteration.

It is worth nothing that K-Means is similar in spirit to coordinate-wise optimization, however, the two "coordinates" we consider, the clustering function $k(i)$, and the collection of prototypes, $m_1, \ldots, m_K$ each accounts for several variables. However, other than this difference the idea is essentially the same.

4.1.3 Soft Clustering – The EM Algorithm

In an iteration of the K-Means algoirithm each point is assigned to the cluster with the nearest prototype. Sometimes this is called "hard" assignment. We explore now an algorithm that assigns the point, in different proportions, to *each* of the clusters — a "soft" assignment. The details of the algorithm involve a good deal of mathematics that is beyond the scope of our course, though the essential ideas are within reach.

In the soft clustering strategy a cluster is no longer described by a single prototype (cluster center), m_k. Rather, in addition to the cluster center we have two additional attributes that describe a cluster. The first of these is a probability, $p(k)$, for the K different clusters: $p(1), \ldots, p(K)$, thus representing the proportion of the points in each cluster.



But there is an additional difference between the soft- and hard-clutering approaches: While in K-Means we used *Euclidean* distance as our measure of distance from the cluster center, now we use a more flexible kind of distance that understands that some directions of "variation" may be priveleged over others. An example of such a situation can be seen in the plot above where the scatter is greater in the "NW-SE" direction than it is in the "NE-SW" direction. If we were to call the lengths of the two arrows in the figure d_1 and d_2, then the squared distance of a

point a from the center, m_k could expressed as

$$D_k^2(a, m_k) = (a_1/d_1)^2 + (a_2/d_2)^2$$

where a_1, a_2 are the coordinates of a point in the "arrows" coordiante system of the figure with origin m_k. Using this asymetric notion of distance, deviations in the "short" direction incease the distance more than devations in the "long" direction. If you remember your Euclidean Geometry, the "constant distance" contours would be ellipses, as in the previous figure.

The reason we are interested in such a distance is that it can be used to describe the likelihood of a point in the cluster. When the distance is low the point is likely, while when the distance is large it is unlikely. This is made more precise in the *conditional probability* for a data point x_i, given a cluster, k:

$$p(x_i|k) = ce^{-\frac{1}{2}D_k^2(x_i,m_k)}$$

Don't worry about the scaling constant c and the exponential in the description of our probability. The important thing is that we have concentric elipses, centered around m_k describing constant probability, while as the probability is lower and lower as we move away from the center, m_k.

The iteration of our soft clustering algorithm is similar in spirit to that of the K-means algorithm. However, we begin not just by intializing the cluster centers, but also the *probabilities* of the clusters, as well as the distance functions for the clusters. For instance, we could begin by choosing the cluster centers randomly, taking the probabilities of the K clusters to be $1/K$, and using regular Euclidean distance for the distance function for each cluster. Having intialized, the soft clustering algorithm iterates the following two steps:

1. For each point we compute the *probability* that it came from each cluster. This is a computation that uses Bayes' rule

$$\begin{aligned} p(k|x_i) &= \frac{p(k)p(x_i|k)}{\sum_{k'} p(k')p(x_i|k')} \\ &= \frac{p(k)e^{-\frac{1}{2}D_k^2(x_i,m_k)}}{\sum_{k'} p(k')e^{-\frac{1}{2}D_{k'}^2(x_i,m_{k'})}} \end{aligned}$$

   where in the 2nd equation we have substituted our definition of the conditional probability in terms of our distance function $D(x_i, m_k)$. We can think of $p(k|x_i)$ as the *proportion* of x_i that is assigned to cluster k.

2. We now reset the cluster parameters $p(k), m_k$, and our distance functions as follows:

$$\begin{aligned} p(k) &= \frac{\sum_i p(k|x_i)}{n} \\ m_k &= \frac{\sum_i p(k|x_i)x_i}{\sum_i p(k|x_i)} \end{aligned}$$

The recalculation of the distance function is a little more complex and less-intuitive for our current course. Besides, we want to save some surprises for your future studies.

The updates we give make intuitive sense with a little thought. $p(k|x_i)$ is the proportion the point x_i we assign to the kth cluster. So if we wanted to get a *count* of the number of points assigned to the kth cluster we would just sum $p(k|x_i)$ over i. Of course we want probabilities in the end, so we divide these counts by the total number of points, n.

Similarly the update for m_k also makes intuitive sense. We have already discussed the computation of a sample as simply the average of the vectors: $m = \frac{1}{n}\sum x_i$. The update above is a minor variation on this idea. To compute m_k we take the proportions of each sample, x_i, assumed to be associated with the k cluster, $p(k|x_i)$, weighting x_i by this proportion. The denominator is just the sum of these proprotions, thus ensuring that we are computing an average — a *weighted* average.

Looking at an implentation of this algorithm is illuminating, so you should consider the gmm_em.r example in the notes. While the soft clustering works, in principle, with *any* number of clusters, we use 3 clusters here for a good reason. The central object of the algorithm is the conditional probabilities, $p(k|x_i)$ that describe our soft assignment

of x_i. If, say, $p(1|x_i) = 1$ then we are certain that x_i belongs to cluster 1; in this case we would color the point x_i as red in the R implementation. Similarly, if $p(2|x_i) = 1$ we would color blue, while if $p(3|x_k) = 1$ we would color green. However, more typically we are unsure of the assignment for x_i so we get some non-zero probabilities for each of $p(1|x_i), p(2|x_i), p(3|x_i)$. In such a case we would *mix* the colors red,blue, and green in coloring x_i. As the iterations of the algorithm evolve you can see points gradually changing from a non-descript brown toward either red, blue, or green. This reflects the increasing confidence the algorithm has in their soft assignments.