$n = \log(1 - c)/\log(c)$

$n = \log(0.05)/\log(0.95)$

$n = -2.9957/-0.0513$

$n = 58.4$

This means that 59 tests of convergence agreeing that the variable is converged establishes the 95% confidence required. (Since “0.4” of a test is impossible, the 58.4 is increased to the next whole number, 59.)

This relationship is used in the demonstration software to test for convergence of variability. Essentially, the method is to keep increasing the number of instances in the sample, checking variability at each step, and wait until the variability is within the error band the required number of times.

Chapter 6: Handling Nonnumerical Variables

Overview

Given the representative sample, as described in Chapter 5, it may well consist of a mixture of variable types. Nonnumerical, or *alpha*, variables present a set of problems different from those of numerical variables. Chapter 2 briefly examined the different types of nonnumerical variables, where they were referred to as nominal and categorical. Distinguishing between the different types of alpha variables is not easy, as they blend into a continuum. Whenever methods of handling alpha variables are used, they must be effective at handling all types across the continuum.

The types of problems that have to be handled depend to some extent on the capabilities, and the needs, of the modeling tools involved. Some tools, such as decision trees, can handle alpha values in their alpha form. Other tools, such as neural networks, can handle only a numeric representation of the alpha value. The miner may need to use several different modeling tools on a data set, each tool having different capabilities and needs. Whatever techniques are used to prepare the data set, they should not distort its information content (i.e., add bias). Ideally, the data prepared for one tool should be useable by any other tool—and should give materially the same results with any tool that can handle the data.

Since all tools can handle numerical data but some tools cannot handle alpha data, the miner needs a method of transforming alpha values into appropriate numerical values.

Chapter 2 introduced the idea that the values in a data set reflect some state of the real world. It also introduced the idea that the ordering of, and spacing between, alpha variables could be recovered and expressed numerically by looking at the data set as a whole. This chapter explores how this can be done. The groundwork that is laid here is needed to cover issues other than just the numeration of alpha values, so rather than covering the same material more than once, several topics are visited, and ideas introduced, in slightly more detail than is needed just for dealing with alpha variables.

Four broad topics are discussed in this chapter. The first is the remapping of variable values, which can apply to both numeric and alpha values, but most often applies to alphas. The miner has to make decisions about the most appropriate representation of alpha variables. There are several automated techniques discussed in this chapter for discovering an appropriate numeric representation of alpha values. Unfortunately, there is no guarantee that these techniques will find even a good representation, let alone the best one. They will find the best numerical representation, given the form in which the alpha is delivered for preparation, and the information in the data set. However, insights and understanding brought by the miner, or by a domain expert, will almost certainly give a much better representation. What must be avoided at all costs is an arbitrary assignment of numbers to alpha labels. The initial stage in numerating alphas is for the miner to replace them with a numeration that has some rationale, if possible.

The second topic is a more detailed look at state space. Understanding state space is needed not only for numerating the alphas, but also for conducting the data survey and for addressing various problems and issues in data mining. Becoming comfortable with the concept of data existing in state space yields insight into, and a level of comfort in dealing with, modeling problems.

The third is *joint frequency distribution tables*. Data sets may be purely numeric, mixed alpha-numeric, or purely alpha. If the data set is purely numeric, then the techniques of this chapter are not needed—at least not for numerating alpha values. Data sets containing exclusively alpha values cannot reflect or be calibrated against associated numeric values in the data set because there are none. Numerating all alpha values requires a different technique. The one described here is based on the frequencies of occurrence of alpha values as expressed in joint frequency distribution tables.

The fourth broad topic is multidimensional scaling (MDS). Chapter 2 also introduced the idea that some alpha variables are most usefully translated into more than one single numeric value (ZIP codes into latitude and longitude, for example; see the explanation in Chapter 2). Taking that idea a step further, some technique is needed to project the values into the appropriate number of dimensions. The technique is multidimensional scaling. Although discussed in this chapter as a means to discover the appropriate dimensionality for a variable, MDS techniques can also be used for reducing the dimensionality of a data set.

6.1 Representing Alphas and Remapping

Exactly how alpha values are best represented depends very much on the needs of the modeling tool. Function-fitting modeling methods are sensitive to the form of the representation. These tools use techniques like regression and neural networks and, to some extent, symbolic regression, evolution programming, and equation evolvers. These tools yield final output that can be expressed as some form of mathematical equation (i.e., *x* = some combination and manipulation of input values). Other modeling methods, such as those sensitive mainly to the ordinality patterns in data, are usually less sensitive, although certainly not entirely insensitive, to representational issues, as they can handle alpha values directly. These include tools based on techniques like some forms of decision trees, decision lists, and some nearest-neighbor techniques.

However, all modeling tools used by data miners are sensitive to the one-to-many problem (introduced in the next section and also revisited in Chapter 11), although there are different ways of dealing with it. The one-to-many problem afflicts numeric variables as well as alpha variables. It is introduced here because for some representations it is an important consideration when remapping the alpha labels. If this problem does exist in an alpha variable, remapping using domain knowledge may prove to be the easiest way to deal with the problem.

There is an empirical way—a rule of thumb—for finding out if any particular remapping is both an improvement and robust. Try it! That is, build a few simple models with various methods. Ensure that at least the remapped values cause no significant degradation in performance over the default choice of leaving it alone. If in addition to doing no harm, it does some good, at least sometimes, it is probably worth considering. This empirical test is true too for the automated techniques described later. They have been chosen because in general, and with much practical testing, they at least do no harm, and often (usually) improve performance depending on the modeling tool used.

In general, remapping can be very useful indeed when one or more of these circumstances is true:

- The information density that can be remapped into a pseudo-variable is high.
- The dimensionality of the model is only modestly increased by remapping.
- A rational, or logical, reasoning can be given for the remapping.
- The underlying rationale for the model requires that the alpha inputs are to be represented without implicit ordering.

6.1.1 One-of-*n* Remapping

Ask a U.S. citizen or longtime resident, “How many states are there?” and you will

probably get the answer “Fifty!” It turns out that for many models that deal with states, there are a lot more. Canadian provinces get squeezed in, as well as Mexican states, plus all kinds of errors. (Where, for instance, is IW?) However, sticking with just the 50 U.S. states, how are these to be represented? In fact, states are usually dealt with quite well by the automated numeration techniques described later in this chapter. However, remapping them is a classic example for neural networks and serves as a good general example of *one-of-n remapping*. It also demonstrates the problems with this type of remapping.

A one-of-*n* representation requires creating a binary-valued pseudo-variable for each alpha label value. For U.S. states, this involves creating 50 new binary pseudo-variables, one for each state. The numerical value is set to “1,” indicating the presence of the relevant particular label value, and “0” otherwise. There are 50 variables, only one of which is “on” at any one time. Only one “on” of the 50 possible, so “one-of-*n*” where in this case “*n*” is 50.

Using such a representation has advantages and disadvantages. One advantage is that the mean of each pseudo-variable is proportional to the proportion of the label in the sample; that is, if 25% of the labels were “CA,” then the mean of the pseudo-variable for the label “CA” would be 0.25. A useful feature of this is that when “State” is to be predicted, a model trained on such a representation will produce an output that is the model’s estimate of the probability of that state being the appropriate choice.

Disadvantages are several:

- Dimensionality is increased considerably. If there are many such remapped pseudo-variables, there can be an enormous increase in dimensionality that can easily prevent the miner from building a useful model.

- The density (in the state example) of a particular state may well be very low. If only the 50 U.S. states were used and each was equally represented, each would have a 2% presence. For many tools, such low levels are almost indistinguishable from 0% for each state; in other words, such low levels mean that no state can be usefully distinguished.

- Again, even when the pseudo-variables have reasonable density for modeling, the various outputs will all be “on” to some degree if they are to be predicted, estimating the probability that their output is true. This allows ranking the outputs for degree of probability. While this can be useful, sometimes it is very unhelpful or confusing to know that there is essentially a 50% chance that the answer is “NY,” and a 50% chance that the answer is “CA.”

6.1.2 *m*-of-*n* Remapping

While the na‹ve one-of-*n* remapping (one state to one variable) may cause difficulties, domain knowledge can indicate very useful remappings that significantly enhance the information content in alpha variables. Since these depend on domain knowledge, they are necessarily situation specific. However, useful remappings for state may include such features as creating a pseudo-variable for “North,” one for “South,” another for “East,” one for “West,” and perhaps others for other features of interest, such as population density or number of cities in the state. This *m*-of-*n* remapping is an advantage if either of two conditions is met. First, if the total number of additional variables is less than the number of labels, then *m*-of-*n* remapping increases dimensionality less than one-of-*n*—potentially a big advantage. Second, if the *m*-of-*n* remapping actually adds useful information, either in fact (by explicating domain knowledge), or by making existing information more accessible, once again this is an advantage over one-of-*n*.

This useful remapping technique has more than one of the pseudo-variables “on” for a single input. In one-of-*n*, one state switched “on” one variable. In *m*-of-*n*, several variables may be “on.” For instance, a densely populated U.S. state in the northeast activates several of the pseudo-variables. The pseudo-variables for “North,” “East,” and “Dense Population” would be “on.” So, for this example, one input label maps to three “on” input pseudo-variables. There could, of course, be many more than three possible inputs. In general, *m* would be “on” of the possible *n*—so it’s called an *m*-of-*n* mapping.

Another example of this remapping technique usefully groups common characteristics. Such character aggregation codings can be very useful. For instance, instead of listing the entire content of a grocery store’s produce section using individual alpha labels in a na‹ve one-of-*n* coding, it may be better to create *m*-of-*n* pseudo-variables for “Fruit,” “Vegetable,” “Root Crop,” “Leafy,” “Short Shelf Life,” and so on. Naturally, the useful characteristics will vary with the needs of the situation. It is usually necessary to ensure that the coding produces a unique pattern of pseudo-variable inputs for each alpha label—that is, for this example, a unique pattern for each item in the produce department. The domain expert must make sure, for example, either that the label “rutabaga” maps to a different set of inputs than the label “turnip,” or that mapping to the same input pattern is acceptable.

6.1.3 Remapping to Eliminate Ordering

Another use for remapping is when it is important that there be no implication of ordering among the labels. The automated techniques described in this chapter attempt to find an appropriate ordering and dimensionality of representation for alpha variables. It is very often the case that an appropriate ordering does in fact exist. Where it does exist, it should be preserved and used. However, it is the nature of the algorithms that they will always find an ordering and some dimensional representation for any alpha variable. It may be that the domain expert, or the miner, finds it important to represent a particular variable without ordering. Using remapping achieves model inputs without implicit ordering.

6.1.4 Remapping One-to-Many Patterns, or Ill-Formed Problems

The one-to-many problem can defeat any function-fitting modeling tool, and many other tools too. The problem arises when one input pattern predicts many output patterns. Since mining tools are often used to predict single values, it is convenient to discuss the problem in terms of predicting a single output value. However, since it is quite possible for some tools to predict several output values simultaneously, throughout the following discussion the single value output used for illustration must be thought of as a surrogate for any more complex output pattern. This is not a problem limited to alpha variables by any means. However, since remapping may provide a remedy for the one-to-many problem, we will look at the problem here.

Many modeling tools look for patterns in the input data that are indicative of particular output values. The essence of a predictive model is that it can identify particular input patterns and associate specific output values with them. The output values will always contain some level of noise, and so a prediction can only be to some degree approximately accurate. The noise is assumed to be "fuzz" surrounding some actual value or range of values and is an ineradicable part of the prediction. (See Chapter 2 for a further discussion of this topic.)

A severe and intractable problem arises when a single input pattern should accurately be associated with two or more discrete output values. Figure 6.1 shows a graph of data points. Modeling these points discovers a function that fits the points very well. The function is shown in the title of the graph. The fit is very good.

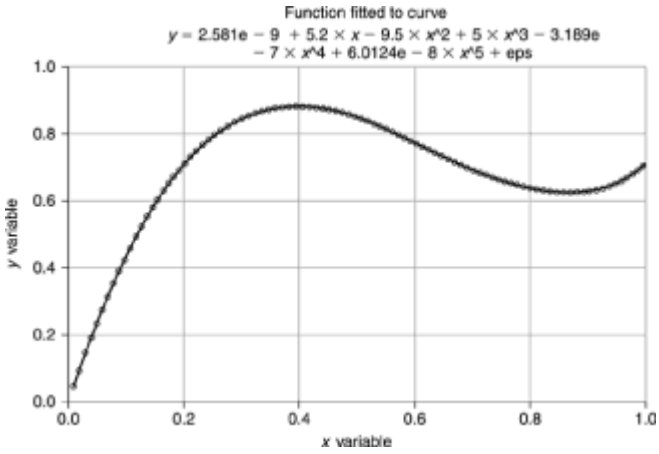

Figure 6.1 The circles show the location of the data points, and the continuous line traces the path of the fitted function. The discovered function fits the function well as there is only a single value of *y* for every value of *x*.

Figure 6.2 shows a totally different situation. Here the original curve has been reflected across the bottom-left, top-right diagonal of the curve, and fitting a function to this curve is a disaster. Why? Because for much of this curve, there is no single value of *y* for every value of *x*. Take the point *x* = 0.7, for example. There are three values of *y: y* = 0.2, *y* = 0.7, and *y* = 1.0. For a single value of *x* there are three values of *y*—and no way, from just knowing the value of *x*, to tell them apart. This makes it impossible to fit a function to this curve. The best that a function-fitting modeling tool can do is to find a function that somehow fits. The one used in this example found as its best approximation a function that can hardly be said to describe the curve very well.

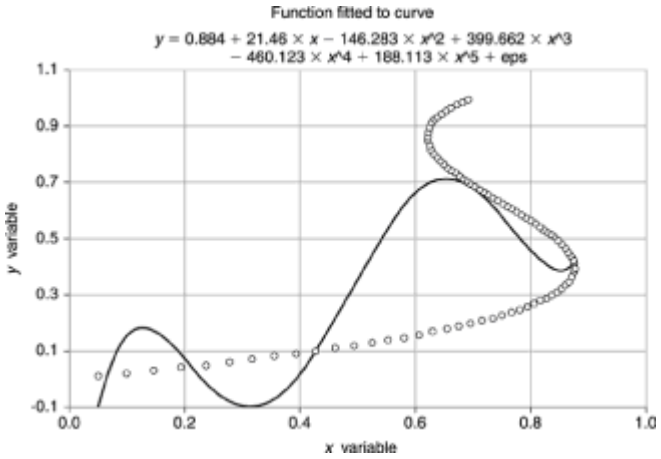

Figure 6.2 The solid line shows the best-fit function that one modeling tool could discover to fit the curve illustrated by the circles. When a curve has multiple predicted (y) values for the input value (*x*), no function can fit the curve.

In Figure 6.2 the input "pattern" (here a single number) is the *x* value. The output pattern is the *y* value. This illustrates the situation in data sets where, for some part of the range, the input pattern genuinely maps to multiple output patterns. One input, many outputs, hence the name one-to-many. Note that the problem is not noise or uncertainty in knowing the value of the output. The output values of *y* for any input values of *x* are clearly specified and can be seen on the graph. It's just that sometimes there is more than one output value associated with an input value. The problem is not that the "true" value lies somewhere between the multiple outputs, but that a function can only give a single output value (or pattern) for a unique input value (or pattern).

Does this problem occur in practice? Do data miners really have to deal with it? The curve shown in Figure 6.1 is a normalized, and for demonstration purposes, somewhat cleaned up, profit curve. The *x* value corresponds to product price, the *y* value to level of profit. As price increases, so does profit for awhile. At some critical point, as price increases, profit falls. Presumably, more customers are put off by the higher price than are offset by the higher profit margin, so overall profit falls. At some point the overall profit rises again with increase in price. Again presumably, enough people still see value in the product at the

higher price to keep buying it so that the increase in price generates more overall profit. Figure 6.1 illustrates the answer to the question What level of profit can I expect at each price level over a range?

Figure 6.2 has price on the *y*-axis and profit on the *x*-axis, and illustrates the answer to the question What price should I set to generate a specific level of profit? The difficulty is that, in this example, there are multiple prices that correspond to some specific levels of profit. Many, if not most, current modeling tools cannot answer this question in the situation illustrated.

There are a number of places in the process where this problem can be fixed, *if* it is detected. And that is a very big if! It is often very hard to determine areas of multivalued output. Miners, when modeling, can overcome the problem using a number of techniques. The data survey (Chapter 11) is the easiest place to detect the problem, if it is not already known to be a problem. However, if it is recognized, and possible, by far the easiest stage in which to correct the problem is during data preparation. It requires the acquisition of some additional information that can distinguish the separate situations. This additional information can be coded into a variable, say, *z*. Figure 6.3 shows the curve in three dimensions. Here it is easy to see that there are unique *x* and *z* values for every point—problem solved!

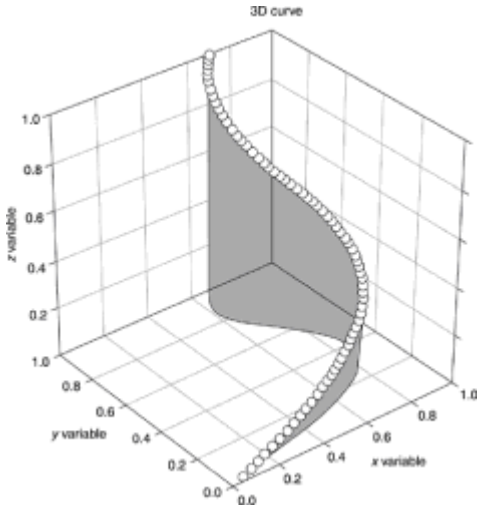

Figure 6.3 Adding a third dimension to the curve allows it to be uniquely characterized by values *x* and *z*. If there is additional information allowing the states to be uniquely defined, this is an easy solution to the problem.

Not quite. In the illustration, the variable *z* varies with *y* to make illustrating the point easy. But because *y* is unknown at prediction time, so is *z*. It's a Catch-22! However, if additional information that can differentiate between the situations is available at preparation time, it is by far the easiest time to correct the problem.

This book focuses on data preparation. Discussing other ways of fixing the one-to-many problem is outside the present book's scope. However, since the topic is not addressed any further here, a brief word about other ways of attacking the problem may help prevent anguish!

There is a clue in the way that the problem was introduced for this example. The example simply reflected a curve that was quite easily represented by a function. If the problem is recognized, it is sometimes possible to alleviate it by making a sort of reflection in the appropriate state space. Another possible answer is to introduce a local distortion in state space that "untwists" the curve so that it is more easily describable. Care must be taken when using these methods, since they often either require the answer to be known or can cause more damage than they cure! The data survey, in part, examines the manifold carefully and should report the location and extent of any such areas in the data. At least when modeling in such an area of the data, the miner can place a large sign "Warning—Quicksand!" on the results.

Another possible solution is for the miner to use modeling techniques that can deal with such curves—that is, techniques that can model surfaces not describable by functions. There are several such techniques, but regrettably, few are available in commercial products at this writing. Another approach is to produce separate models, one for each part of the curve that is describable by a function.

6.1.5 Remapping Circular Discontinuity

Historians and religions have debated whether time is linear or circular. Certainly scientific time is linear in the sense that it proceeds from some beginning point toward an end. For miners and modelers, time is often circular. The seasons roll endlessly round, and after every December comes a January. Even when time appears to be numerically labeled, usually ordinally, the miner should consider what nature of labeling is required inside the model.

Because of the circularity of time, specifying timelike labels has particular problems. Numbering the weeks of the year from "1" to "52" demonstrates the problem. Week 52, on a seasonal calendar, is right next to week 1, but the numbers are not adjacent. There is discontinuity between the two numbers. Data that contains annual cycles, but is ordered as consecutively numbered week labels, will find that the distortion introduced very likely prevents a modeling tool from discovering any cyclic information.

A preferable labeling might set midsummer as "1" and midwinter as "0." For 26 weeks the "Date" flag, a *lead variable*, might travel from "0" toward "1," and for the other 26 weeks from "1" toward "0." A *lag variable* is used to unambiguously define the time by reporting what time it was at some fixed distance in the past. In the example illustrated in Figure 6.4, the lag variable gives the time a quarter of a year ago. These two variables provide an unambiguous indication of the time. The times shown are for solstices and equinoxes,

but every instant throughout the cycle is defined by a unique pair of values. By using this representation of lead and lag variables, the model will be able to discover interactions with annual variations.

Figure 6.4 An annual “clock.” The time is represented by two variables—one showing the time now and one showing where the time was a quarter of a year ago.

Annual variation is not always sufficient. When time is expected to be important in any model, the miner, or domain expert, should determine what cycles are appropriate and expected. Then appropriate and meaningful continuous indicators can be built. When modeling human or animal behavior, various-period circadian rhythms might be appropriate input variables. Marketing models often use seasonal cycles, but distance in days from or to a major holiday is also often appropriate. Frequently, a single cyclic time is not enough, and the model will strongly benefit from having information about multiple cycles of different duration.

Sometimes the cycle may rise slowly and fall abruptly, like “weeks to Thanksgiving.” The day after Thanksgiving, the effective number of weeks steps to 52 and counts down from there. Although the immediately past Thanksgiving may be “0” weeks distant, the salient point is that once “this” Thanksgiving is past, it is immediately 52 weeks to next Thanksgiving. In this case the “1” through “52” numeration is appropriate—but it must be anchored at the appropriate time, Thanksgiving in this case. Anchoring “weeks to Thanksgiving” on January 1st, or Christmas, say, would considerably reduce the utility of the ordering.

As with most other alpha labels, appropriate numeration adds to the information available for modeling. Inappropriate labeling at best makes useful information unavailable, and at worst, destroys it.

6.2 State Space

State space is a space exactly like any other. It is different from the space normally perceived in two ways. First, it is not limited to the three dimensions of accustomed space (or four if you count time). Second, it can be measured along any ordered dimensions that are convenient.

For instance, choosing a two-dimensional state space, the dimensions could be “inches of rain” and “week of the year.” Such a state space is easy to visualize and can be easily drawn on a piece of paper in the form of a graph. Each dimension of space becomes one of the axes of the graph. One of the interesting things about this particular state space is that, unlike our three-dimensional world, the values demarking position on a dimension are bounded; that is to say, they can only take on values from a limited range. In the normal three-dimensional world, the range of values for the dimensions “length,” “breadth,” and “height” are unlimited. Length, breadth, or height of an object can be any value from the very minute—say, the Planck constant (a very minute length indeed)—to billions of light-years. The familiar space used to hold these objects is essentially unlimited in extent.

When constructing state space to deal with data sets, the range of dimensional values is limited. Modeling tools do not deal with monotonic variables, and thus these have to be transformed into some reexpression of them that covers a limited range. It is not at all a mathematical requirement that there be a limit to the size of state space, but the spaces that data miners experience almost always are limited.

6.2.1 Unit State Space

Since the range of values that a dimension can take on are limited, this also limits the “size” of the dimension. The range of the variable fixes the range of the dimension. Since the limiting values for the variables are known, all of the dimensions can be *normalized*. Normalizing here means that every dimension can be constructed so that its maximum and minimum values are the same. It is very convenient to construct the range so that the maximum value is 1 and the minimum 0. The way to do this is very simple. (Methods of normalizing ranges for numeric variables are discussed in Chapter 7.)

When every dimension in state space is constructed so that the maximum and minimum values for each range are 1 and 0, respectively, the space is known as *unit state space*—“unit” because the length of each “side” is one unit long; “state space” because each uniquely defined position in the space represents one particular *state* of the system of variables. This transformation is no more than a convenience, but making such a transformation allows many properties of unit state space to be immediately known. For instance, in a two-dimensional unit state space, the longest straight line that can be constructed is the corner-to-corner diagonal. State space is constructed so that its dimensions are all at right angles to each other—thus two-dimensional state space is

rectangular. Two-dimensional unit state space not only is rectangular, but has “sides” of the same unit length, and so is square. Figure 6.5 shows the corner-to-corner diagonal line, and it immediately is clear that that the Pythagorean theorem can be used to find the length of the line, which must be 1.41 units.

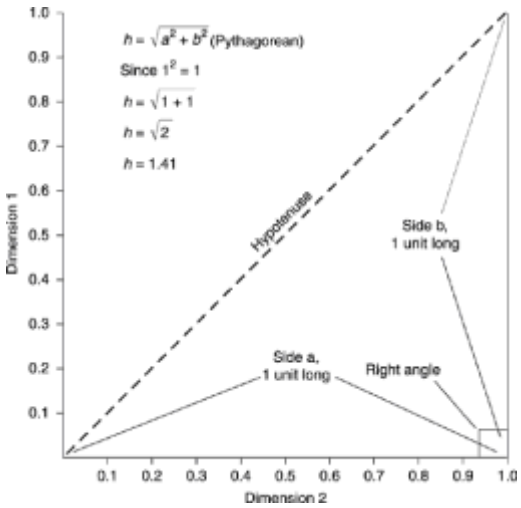

Figure 6.5 Farthest possible separation in state space.

6.2.2 Pythagoras in State Space

Two-dimensional state space is not significantly different from the space represented on the surface of a piece of paper. The Pythagorean theorem can be extended to a three-dimensional space, and in a three-dimensional unit state space, the longest diagonal line that can be constructed is 1.73 units long. What of four dimensions? In fact, there is an analog of the Pythagorean theorem that holds for any dimensionality of state space that miners deal with, regardless of the number of dimensions. It might be stated as: In any right-angled multiangle, the square on the multidimensional hypotenuse is equal to the sum of the squares on all the other sides. The length of the longest straight line that can be constructed in a four-dimensional unit state space is 2, and of a five-dimensional unit state space, 2.24. It turns out that this is just the square root of the number of sides, since the square on a unit side, the square of 1, is just 1.

This means that as more dimensions are added, the longest straight line that can be drawn increases in length. Adding more dimensions literally adds more space. In fact, the longest straight line that can be drawn in unit state space is always just the square root of the number of dimensions.

6.2.3 Position in State Space

Instead of just finding the longest line in state space, the Pythagorean theorem can be used to find the distance between any two points. The position of a point is defined by its

coordinates, which is exactly what the instance values of the variables represent. Each unique set of values represents a unique position in state space. Figure 6.6 shows how to discover the distance between two points in a two-dimensional state space. It is simply a matter of finding the distance between the points on one axis, and then on the other axis, and then the diagonal length between the two points.

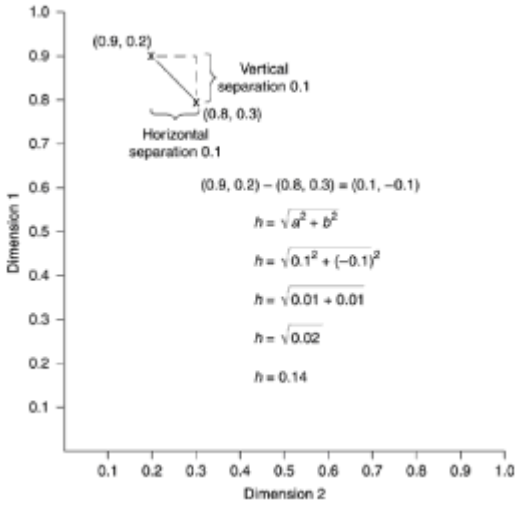

Figure 6.6 Finding the distance between two points in a 2D state space.

Just as with finding the length of the longest straight line that can be drawn in state space, so too this finding of the distance between two points can be generalized to work in higher-dimensional state spaces. But each point in state space represents a particular state of the system of variables, which in turn represent a particular state of the object or event existing in the real world that was being measured. State space provides a standard way of measuring and expressing the distance between any states of the system, whether events or objects.

Using unit state space provides a frame of reference that allows the distance between any two points in that space to be easily determined. Adding more dimensions, because it adds more space in which to position points, actually moves them apart. Consider the points shown in Figure 6.6 that are 0.1 units apart in both dimensions. If another dimension is added, unless the value of the position on that dimension is identical for both points, the distance between the points increases. This is a phenomenon that is very important when modeling data. More dimensions means more sparsity or distance between the data points in state space. A modeling tool has to search and characterize state space, and too many dimensions means that the data points disappear into a thin mist!

6.2.4 Neighbors and Associates

Points in state space that are close to each other are called *neighbors*. In fact, there is a data modeling technique called “nearest neighbor” or “*k*-nearest neighbor” that is based on this concept. This use of neighbors simply reflects the idea that states of the system that are close together are more likely to share features in common than system states further apart. This is only true if the dimensions actually reflect some association between the states of the system indicated by their positions in state space.

Consider as an example Figure 6.7. This shows a hypothetical relationship in two-dimensional unit state space between human age and height. Since height changes as people grow older up to some limiting age, there is an association between the two dimensions. Neighbors close together in state space tend to share common characteristics up to the limiting age. After the limiting age—that is, the age at which humans stop growing taller—there is no particular association between age and height, except that this range has lower and upper limits. In the age dimension, the lower limit is the age at which growth stops, and the upper limit is the age at which death occurs. In the height dimension, after the age at which growth stops, the limits are the extremes of adult height in the human population. Before growth stops, knowing the value of one dimension gives an idea of what the value of the other dimension might be. In other words, the height/age neighborhood can be usefully characterized. After growth stops, the association is lost.

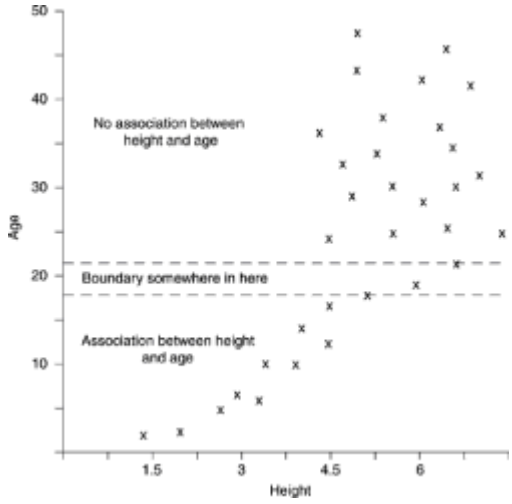

Figure 6.7 Showing the relationship between neighbors and association when there is, and is not, an association between the variables.

This simplified example is interesting because although it is simplified, it is similar to many practical data characterization problems. For sets of variables other than just human height and weight, the modeler might be interested in discovering that there are boundaries. The existence and position of such boundaries might be an unknown piece of information. The changing nature of a relationship might have to be discovered. It is clear that for some part of the range of the data in the example, one set of predictions or

inferences can be made, and in another part of the same data set, wholly different inferences or predictions must be made. This change in the nature of the neighborhood from place to place can be very important. In two dimensions it is easy to see, but in high-dimensionality spaces this can be difficult to discover.

6.2.5 Density and Sparsity

Before continuing, a difference in the use of the terms *location* or *position*, and *points* or *data points*, needs to be noted.

In any space there are an infinite number of places or positions that can be specified. Even the plane represented by two-dimensional state space has an infinite number of positions on it that can be represented. In fact, even on a straight line, between any two positions there are an infinite number of other positions. This is because it is always possible to specify a location on a dimension that is between any two other locations. For instance, between the locations represented by 0.124 and 0.125 are other locations represented by 0.1241, 0.1242, 0.1243, and so on. This is a property of what is called the *number line*. It is always possible to use more precision to specify more locations. The terms *location* or *position* are used to represent a specific place in space.

Data, of course, has values—instance values—that can be represented as specifying a particular position. The instance values in a data set, representing particular states of the system, translate into representing particular positions in state space. When a particular position is actually represented by an instance value, it is referred to as a *data point* or *point* to indicate that this position represents a measured state of the system.

So the terms *location* and *position* are used to indicate a specific set of values that might or might not be represented by an instance value in the data. The terms *point* and *data point* indicate that the location represents recorded instance values and therefore corresponds to an actual measured state of the system.

Turning now to consider density, in the physical world things that are dense have more “stuff” in them per unit volume than things that are less dense. So too, some areas of state space have more data points in them for a given volume than other areas. State space density can be measured as the number of data points in a specific volume. In a dense part of state space, any given location has its nearest neighboring points packed around it more closely than in more sparsely populated parts of state space.

Naturally, in a state space of a fixed number of dimensions, the absolute mean density of the data points depends on the number of data points present and the size of the space. The number of dimensions fixes unit state space volume, but the number of data points in that volume depends only on how much data has been collected. However, given a representative sample, if there are associations among the dimensions, the relative density of one part of state space remains in the same relationship to the relative density

of another part of the same space regardless of how many data points are added.

If this is not intuitive, imagine two representative samples drawn from the same population. Each sample is projected into its own state space. Since the samples are representative of the same population, both state spaces will have the same dimensions, normalized to the same values. If this were not so, then the samples would not be truly representative of the same population. Since both data sets are indeed representative of the same population, the distributions of the variables are, for all practical purposes, identical in both samples, as are the joint distributions. Thus, any given specific area common to both state spaces will have the same proportion of the total number of points in each space—not necessarily the same actual number of points, as the representative samples may be of different sizes, but the same relative number of points.

Because both representative data sets drawn from a common population have similar relative density throughout, adding them together—that is, putting all of the data points into a common state space—does not change the relative density in the common state space. As a specific example, if some defined area of both state spaces has a relative density twice the mean density, when added together, the defined area of the common state space will also have a density twice the mean—even though the mean will have changed. Table 6.1 shows an example of this.

TABLE 6.1 State space density.

	Mean density	Specific area density
Sample 1	20	40
Sample 2	10	20
Combined	30	60

This table shows the actual number of data points in two samples representative of the same population. The specific area density in each sample is twice the mean density even though the number of points in each sample is different. When the two samples are combined, the combined state space still has the same relative specific area density as each of the original state spaces. So it is that when looking at the density of a particular volume of space, it is *relative density* that is most usefully examined.

There are difficulties in determining density just by looking at the number of points in a given area, particularly if in some places the given volume only has one data point, or even no data points, in it. If enough data points from a representative sample are added, eventually any area will have a measurable density. Even a sample of empty space has some density represented by the points around it. The density at any position also depends on the size and shape of the area that is chosen to sample the density. For many purposes this makes it inconvenient to just look at chunks of state space to estimate density.

Another way of estimating density is to choose a point, or a position, and estimate the distance from there to each of the nearest data points in each dimension. The mean distance to neighboring data points serves as a surrogate measurement for density. For many purposes this is a more convenient measure since every point and position then has a measurable density. The series of illustrations in Figure 6.8 shows this. The difficulty, of course, is in determining exactly what constitutes a nearest neighbor, and how many to use.

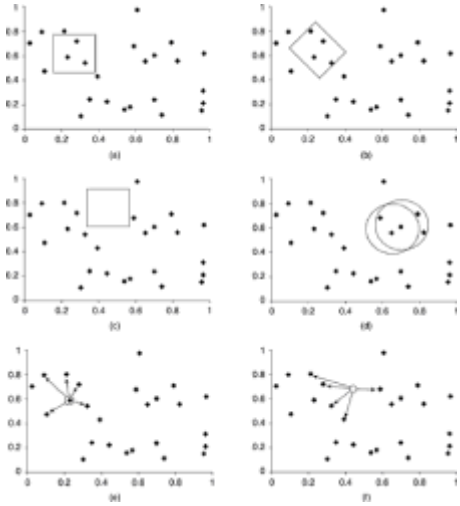

Figure 6.8 Estimating density: inside a square (a), rotating the same square (b), same square moved to an unoccupied area (c), circular area (d), distance to a number of neighbors (e), and distance to neighboring points from an empty position (f).

Figure 6.8(a) shows the density inside a square to be 3. The same square in the same location but rotated slightly could change the apparent density, as shown in Figure 6.8(b). Figure 6.8(c) shows a square in an unoccupied space, which makes deciding what the density is, or what it could be if more points were added, problematic. Using a circular area can still have large jumps in density with a small shift in position, as shown in Figure 6.8(d). Figure 6.8(e) shows that measuring the distance to a number of neighbors gives each point a unique density. Even an empty position has a measurable density by finding the distances to neighboring points, as shown in Figure 6.8(f).

A better way of estimating density determines a weighted distance from every point in state space to every other point. This gives an accurate density measure and produces a continuous density gradient for all of space. Determining the nature of any location in space uses the characteristics of every point in space, weighted by their distance. This method allows every point to “vote” on the characteristics of some selected location in space according to how near they are, and thus uses the whole data set. Distant points have little influence on the outcome, while closer points have more influence. This works well for determining nature and density of a point or location in state space, but it does necessitate that any new point added requires recalculation of the entire density structure. For highly populated state spaces, this becomes computationally intensive (slow!).

6.2.6 Nearby and Distant Nearest Neighbors

As with many things in life, making a particular set of choices has trade-offs. So it is with nearest-neighbor methods. The first compromise requires deciding on the number of nearby neighbors to actually look at. Figures 6.8(e) and 6.8(f) illustrate five neighbors near to a point or position. Using nearest neighbors to determine the likely behavior of the system for some specified location has different needs than using nearest neighbors to estimate density. When estimating system behavior, using some number of the closest neighbors in state space may provide the best estimate of system behavior. It usually does not provide the best estimate of density.

Figure 6.9(a) illustrates why this might be the case. This example shows the use of four neighbors. The closest neighbors to the point circled are all on one side of the point. Using only these points to estimate density does not reflect the distance to other surrounding points. A more intuitive view of density requires finding the nearest neighbors in “all directions” (or omnidirectionally) around the chosen point. Having only the very closest selected number of points all biased in direction leads to an overestimate of the omnidirectional density.

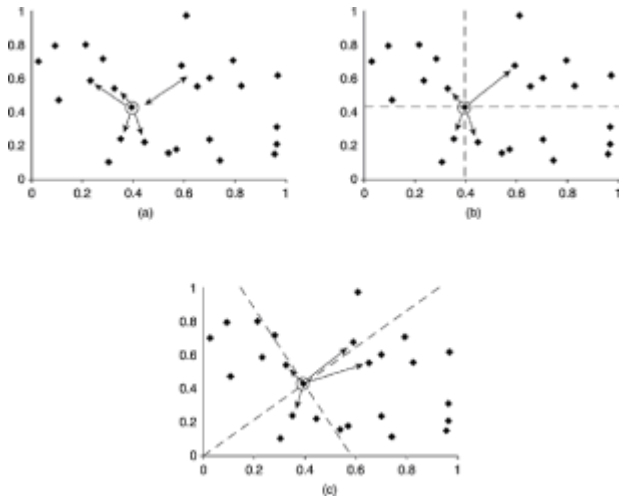

Figure 6.9 Results of estimating density with nearest neighbors: overestimate (a), better estimate by looking for nearest neighbors in specific areas (b), and change in estimate by rotating the axes of same specific areas (c).

One way around this shortcoming is to divide up the area to be searched, and to find a nearest neighbor in each division, as shown in Figure 6.9(b). Still using four neighbors, dividing space into quadrants and finding a nearest neighbor in each quadrant leads to a better estimate of the omnidirectional density. However, no compromise is perfect. As Figure 6.9(c) shows, rotating the axes of the quadrants can significantly change the estimated density.

Since “divide and conquer” provides useable estimates of density and serves to identify nearest neighbors, the demonstration code uses this both for neighbor estimation and as a quick density estimation method.

6.2.7 Normalizing Measured Point Separation

Using normalized values of dimensions facilitates building a *unit* state space. This has some convenient properties. Can distance measured between points be normalized? State space size (volume) is proportional to the number of dimensions that constitute the space. In a unit state space, the maximum possible separation between points is known—the square root of the number of dimensions. Regardless of the number of dimensions, no two points can be further separated than this distance. Similarly, no two positions can be closer together than having no separation between them. This means that the separation between points *can* be normalized. Any particular separation can be expressed as a fraction of the maximum possible separation, which comes out as a number between 0 and 1.

Density is not usually usefully expressed as a normalized quantity. Since it is relative density that is of interest, density at a point or location is usually expressed relative to the mean, or average, density. It is always possible for a particular position to be any number of times more or less dense than the average density, say, 10 or 20 times. It is quite possible to take the maximum and minimum density values found in a particular state space and normalize the range, but is it usually more useful to know the density deviation from the mean value. In any case, as more data points are added, the maximum, minimum, and mean values will change, requiring recalibration if density is to be normalized. However, as discussed above, given a representative sample data set, relative density overall will not change with additional data from the same population.

6.2.8 Contours, Peaks, and Valleys

Instead of simply regarding the points in state space as having a particular density, imagine that the density value is graduated across the intervening separation. Between a

point of high density and its lower-density neighbor, the density decreases across the distance from high value to low. State space can be imagined as being permeated by a continuous gradient of density, perhaps going “down” toward the densest areas, and “up” toward the least dense areas. This up-and-down orientation conjures up the idea of a surface of some sort that represents the expression of the gradient. The surface has high points representing areas of least density and low points representing areas of most density. The slope of the surface represents the rate of change in density at that position.

Three-dimensional surfaces of this sort, surfaces such as that of the earth’s, can be mapped topographically. Such maps often show lines that are traced over the surface marking the positions of a particular constant elevation. Such lines are known as *contours*. Other sorts of contours can be traced, for example, along a ridge between two high points, or along the deepest part of a valley between two low points. A density surface can also be mapped with a variety of contours analogous to those used on a topographic map.

6.2.9 Mapping State Space

Exploring features of the density surface can reveal an enormous amount of useful, even vital information. Exploring the density map forms a significant part of the data survey. For example, tracing all of the “ridges”—that is, tracing out the contours that wend their way through the least densely populated areas of state space—leads to identifying groups of natural clusters. Each cluster of points swarms about a point of maximum density. Keeping in mind that this map ultimately represents some state of an object in the real world, the mapped clusters show the systems’ “preferred” states—or do they? Maybe they show evidence of bias in the data collection. Perhaps data about those states was for some reason preferentially collected, and they predominate simply because they were the easiest to collect. (Chapter 11 covers the practical application of this more fully. Here we are just introducing the ideas that will be used later.)

6.2.10 Objects in State Space

Sometimes a more useful metaphor for thinking of the points in state space is as a geometric object of some sort, even though when more than three dimensions are used it is hard to imagine such an object. Nonetheless, if the points in state space are thought of as “corners,” it is possible to join them with the analogs of lines and surfaces.

Three points in a two-dimensional state space could form the corners of a triangle. To construct the triangle, the points are simply joined by lines. Similarly, in three-dimensional space, points are joined by planes to form three-dimensional figures.

An interesting feature of the object analogy is that, just as with objects in familiar space, they can cast “shadows.” In the familiar world, a three-dimensional object illuminated by the sun casts a two-dimensional shadow. The shadow represents a more or less distorted

image of the object. So it is in state space that higher-dimensional objects can cast lower-dimensional shadows. This ability of objects to cast shadows is one of the features used in multidimensional scaling.

6.2.11 Phase Space

Phase space is identical to state space in almost all respects, with a single exception. *Phase space* is used to represent features of objects or systems *other* than their state. Since a system state is not represented in phase space, the name of the space changes to reflect that. The reason to introduce what is essentially an identical sort of space to state space is that when numerating alpha values, a space is needed in which to represent the distances between the *labels*. Alpha labels, you will recall, do not represent states of the system, but values of a particular variable. In order to numerate alpha labels, or in other words to assign them particular numeric values indicating their order and spacing, a space has to be created in which the labels can exist. The alpha labels are arrayed in this space, each with a particular distance and direction from its neighboring labels. Finding the appropriate place to put the labels in phase space is discussed in the next section. The point is that when the appropriate positions for the labels are known, then the appropriate label values can be found.

The most important point to note here is that the name of the space does not change its properties. It simply identifies if the space is used to hold states of a system of variables (state space) or some other features (phase space).

Why the name “phase space”? Well, “phase” indicates a relationship between things. Electrical engineers are familiar with three-phase alternating-current power. This only means that three power pulses occur in a complete cycle, and that they have a specific, fixed relationship to each other. As another example, the phases of the moon represent specific, and changing, relationships between the earth, moon, and sun. So too with phase space. This is an imaginary space, identical in almost all respects to state space, except that relationships, or phases, between things are represented.

6.2.12 Mapping Alpha Values

So far, all of the discussion of state space has assumed dimensions that are numerically scaled and normalized into the range 0 to 1. Where do alpha values fit in here?

Between any two variables, whether alpha or numeric, there is some sort of relationship. As in the height/age example, characterizing the precise nature of the relationship may be difficult. In some parts of the range, the variables may allow better or worse inferences about how the values relate. Nonetheless, it is the existence of a relationship that allows any inferences to be made. Statistically, the variables may be said to be more or less independent of each other. If fully independent, it could be said that there is no relationship. Actually, it is more accurate to say that when variables are independent,

knowing something about the state of one variable tells nothing about the state of the other. There is still a relationship, but it carries no useful information. As an example of complete statistical independence, flipping a coin and knowing the result tells you nothing whatever about the time at which the flip was made.

The system of variables that is used to populate state space is exactly that, a system. A system has interreacting and interlocking components. The system reflects, more or less, the real world, and the world is not a purely random phenomenon. The instance values represent snapshots of parts of the system in action. It may be that the system is not well understood; indeed, it may be that understanding the system is the whole purpose of the data exploration enterprise. Nonetheless, a system is not going to have all of its components independent of each other. If all of the components have no relation whatsoever to each other, it hardly qualifies as a system!

It is the interrelationship between the alpha values and the system of variables as a whole that allows their appropriate numeration. Numeration does not recover the actual values appropriate for an alpha variable, even if there are any. It may very well be that there are no inherently appropriate actual values. Although cities, for instance, can be ranked (say, through an opinion poll) for “quality of life,” placed in order, and separated by an appropriate distance along the ranking, there is no absolute number associated with each position. The quality-of-life scale might be from 1 to 10, or 1 to 100, or 0 to 1. It could even be from 37.275 to 18.462, although that would not be intuitive to humans. What is recoverable is the appropriate order and separation. For convenience, the scale for recovery is normalized from 0 to 1, which allows them to be conveniently positioned in unit state space.

6.2.13 Location, Location, Location!

In real estate, location is all. So too when mapping alphas. The points in state space can be mapped. Alpha variables that are in fact associated with the system of variables can also be appropriately placed on this map. The values of an alpha variable are labels. The numeration method associates each label with some appropriate particular “area” on the state space map. (It is an area in two dimensions, a volume in three dimensions, and some unnamed analog in more than three. For convenience it is referred to throughout this explanation as an “area.”) Discovering the appropriate location of the area is the heart of the method; having done this, the problem then is to turn the high-dimensionality position into an appropriate number. The techniques for doing that are discussed later in this chapter in the section on multidimensional scaling.

The simplest state space that can contain two variables is a two-dimensional state space. If one of the variables is numeric and one alpha, the problem of finding an appropriate value from multiple numeric dimensions does not exist since there is only a single dimension of numeric value (which means only one number) at any location. While a single numeric may not provide a particularly robust estimation of appropriate numeration

of alphas, it can provide an easily understood example.

6.2.14 Numerics, Alphas, and the Montreal Canadiens

Table 6.2 shows a list of the team members on the 1997/1998 roster, together with their height and weight. There is an alpha variable present in this data set—"Position." Unfortunately, if used as an example, when finished there is no way to tell if appropriate numerical values are assigned since the labels have no inherent ordering. With no inherent ordering to compare the recovered values against, the results cannot be checked. A convincing first example needs to be able to be checked for accuracy! So, for the purpose of explanation, a numerical variable will be labeled with alpha labels. Then, when values have been "recovered" for these labels, it is easy to compare the original values with those recovered to see if indeed an appropriate ordering and spacing have been found. With the underlying principles visible by using an example that numerates labels derived from what is actually a numeric variable, we can examine the problem of numerating "Position" as a second example.

TABLE 6.2 Montreal Canadiens roster in order of player weight.

Position	Num	Name	Height	Weight	Code	DoB	NmHt
Defense	34	Peter Popovic	6.5	235	a	10-Feb-68	1
Defense	38	Vladimir Malakhov	6.3	227	b	30-Aug-68	0.759
Forward	21	Mick Vukota	6.08	225	c	14-Sep-66	0.494
Forward	23	Turner Stevenson	6.25	220	d	18-May-72	0.699
Defense	22	Dave Manson	6.17	219	e	27-Jan-67	0.602
Forward	24	Scott Thornton	6.25	219	e	9-Jan-71	0.699
Forward	44	Jonas	6.25	215	f	29-Aug-72	0.699

		Hoglund					
Defense	5	Stephane Quintal	6.25	215	f	22-Oct-68	0.699
Defense	33	Zarley Zalapski	6.08	215	f	22-Apr-68	0.494
Forward	37	Patrick Poulin	6.08	210	g	23-Apr-73	0.494
Reserve	55	Igor Ulanov	6.08	205	h	1-Oct-69	0.494
Forward	26	Martin	6.08	205	h	11-Mar-71	0.494
		Rucinsky					
Defense	43	Patrice Brisebois	6.17	204	j	27-Jan-71	0.602
Forward	28	Marc Bureau	6.08	202	k	19-May-66	0.494
Forward	27	Shayne Corson	6.08	199	m	13-Aug-66	0.494
Defense	52	Craig Rivet	6.17	195	n	13-Sep-74	0.602
Forward	17	Benoit Brunet	6	194	p	24-Aug-68	0.398
Forward	49	Brian Savage	6.08	191	q	24-Feb-71	0.494
Forward	25	Vincent Damphousse	6.08	191	q	17-Dec-67	0.494
Forward	71	Sebastien Bordeleau	5.92	188	r	15-Feb-75	0.301
Forward	15	Eric Houde	5.83	186	s	19-Dec-76	0.193

Forward	8	Mark Recchi	5.83	185	t	1-Feb-68	0.193
Defense	29	Brett Clark	6	182	u	23-Dec-76	0.398
Reserve	11	Saku Koivu	5.83	182	u	23-Nov-74	0.193
Goal	35	Andy Moog	5.67	177	v	18-Feb-60	0
Goal	41	Jocelyn Thibault	5.92	170	w	12-Jan-75	0.301

Example 1—A Weighty Problem

For the convenience of the example, the weights of the athletes are labeled from “a” through “w,” missing out those letters that might be confused with numbers like “l” and “o.” To make it easier to see what is happening, “a” represents the heaviest weight and “w” the lightest. The labels could have been assigned arbitrarily; the ordering only helps to show what is happening. (Note: It causes a problem to assign numeric values to alpha labels arbitrarily, *not* vice versa. Alpha labels are, by nature, arbitrary.) The numeric variable used in this example will be “Height.” To see how well the normalized weights can be recovered from alpha labels, the weight will be converted to an alpha value. The actual weights can be compared with the recovered values to determine if the method was effective.

Table 6.2 shows the names, heights, and weights of the athletes. The heights, the numeric variable in this example, are shown in feet and decimal fractions of a foot. In order to construct a unit state space, height has to be normalized, and this is also shown. Next are the weights in pounds and their associated labels. Since some of the athletes weigh the same amount, these weights are assigned identical labels.

The athletes’ heights form a one-dimensional state space, which can be easily represented by an ordered list such as the one in Table 6.3. The column on the left shows the ordered, normalized heights for each athlete, and the right-hand column shows the alpha (weight) labels. Some of the labels appear more than once for those athletes having similar weights. When “projecting” the height values onto the weight labels, since there is only a single numeric dimension, the values of most of the labels are simply the normalized height values. Where there are multiple occurrences of labels, the average of

the normalized values is taken. Table 6.4 shows this.

TABLE 6.3 Normalized heights and weight code. Some codes, such as "f," are duplicated.

NmHt	WC
1	a
0.759	b
0.699	d
0.699	e
0.699	f
0.699	f
0.602	e
0.602	j
0.602	n
0.494	c
0.494	f
0.494	g
0.494	h
0.494	h
0.494	k
0.494	m
0.494	q

0.494	q
0.398	p
0.398	u
0.301	r
0.301	w
0.193	s
0.193	t
0.193	u
0	v

TABLE 6.4 Values of the weight codes.

Weight code	Code numeration
a	1
b	0.76
c	0.49
d	0.7
e	0.65
f	0.7
g	0.49
h	0.49

j	0.6
k	0.49
m	0.49
n	0.6
p	0.4
q	0.49
r	0.3
s	0.19
t	0.4
u	0.19
v	0
w	0.3

Since this simple example has only a single numeric dimension, the appropriate values are simply the single-dimensional representation in the table. Multidimensional examples are reduced using the multidimensional scaling techniques discussed later. Note that this example does not say that the weight labels are assigned the height values. It says instead that the appropriate normalized numeric value for a weight label is the matching normalized height value. (True in this case because there is only a single numeric variable in the system.)

Does this work? Figure 6.10 shows a graph of the actual normalized weights and the "recovered" normalized values for the labels. The fit is quite good. The correlation coefficient is about 0.85, where 0 indicates no predictive relationship at all, and 1 indicates a perfectly predictive relationship. Since this is a very small sample on only one numeric variable, this is a reasonable fit. It certainly isn't perfect, but it provides a reasonable and useful recovery of the appropriate weight label values and intervals. Naturally, with a larger sample, and with a true system of variables to draw upon, better mappings of numerating alpha values can be achieved.

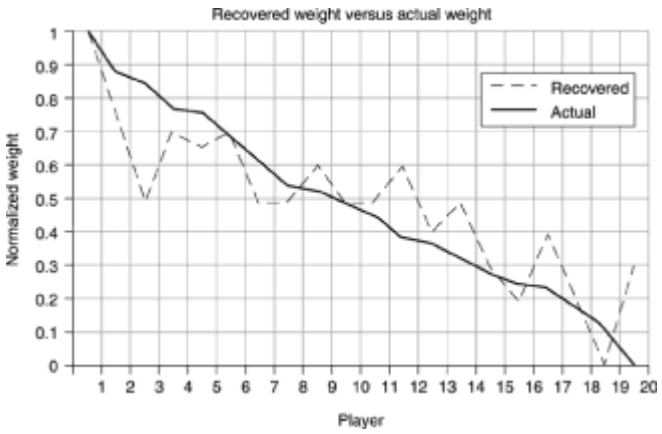

Figure 6.10 Plot of actual weights for Montreal Canadiens versus "recovered" weights. The fit is moderately good, correlation 0.85, certainly better than arbitrary assignment of numbers to the labels.

Example 2—Player Position

The variable "Position" is inherently an alpha variable. That is to say, it has no apparent inherent numeric valuation. It is exactly this sort of variable that requires appropriate numeration, and it is for these types of variables that numeration techniques are needed.

For ease of explanation, the variable "Position" will be numerated on a two-dimensional state space built from the normalized values of "Height" and "Weight."

Plotting all of the height/weight positions shown in Table 6.5 in the state space shows the pattern, or "shape," that each of the "Positions" makes. These positions are shown in Figure 6.11. Each of these shapes is summarized by finding its "center." There are several ways of finding a pattern's central location. One easy method is to find the average (mean) of the values of each label for each dimension.

TABLE 6.5 Position, normalized heights, and normalized weights for Montreal Canadiens.

Position	Height	Weight	Position	Height	Weight
Defense	1.0000	1.0000	Forward	0.4940	0.5385
Defense	0.7590	0.8769	Forward	0.4940	0.4923

Forward	0.6988	0.7692	Forward	0.4940	0.4462
Forward	0.6988	0.7538	Forward	0.4940	0.3231
Forward	0.6988	0.6923	Forward	0.4940	0.3231
Defense	0.6988	0.6923	Forward	0.3976	0.3692
Defense	0.6024	0.7538	Defense	0.3976	0.1846
Defense	0.6024	0.5231	Forward	0.3012	0.2769
Defense	0.6024	0.3846	Goal	0.3012	0.0000
Forward	0.4940	0.8462	Forward	0.1928	0.2462
Defense	0.4940	0.6923	Forward	0.1928	0.2308
Forward	0.4940	0.6154	Reserve	0.1928	0.1846
Reserve	0.4940	0.5385	Goal	0.0000	0.1077

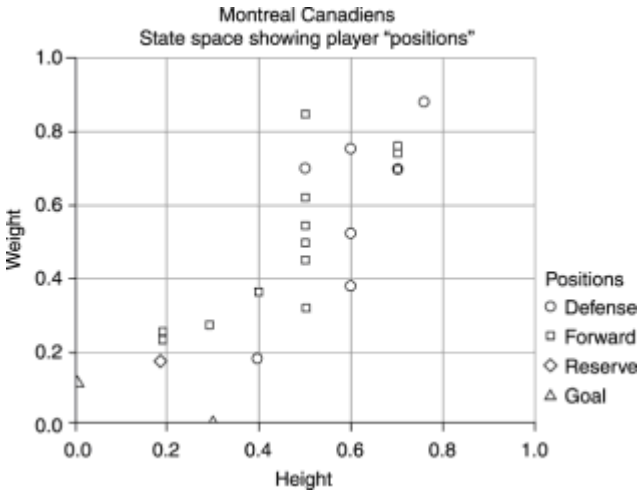

Figure 6.11 The normalized values of height and weight for each player are plotted in 2D state space. Each position type is identified. Taking the values for each alpha label type together, their outline covers an area of state space.

Using the Shape centers from Table 6.6, which are the central positions for each value

(label) of the variable “Position,” the variable Shape can be laid over the state space. The centers and Shape are shown in Figure 6.12. The Shape in this figure seems to be close to a straight line. Still, the points do not fall exactly on a straight line, and converting this Shape into normalized values is discussed in the section about multidimensional scaling, later in this chapter. The Shape discovered in state space is taken, placed into, and manipulated in a separate phase space.

TABLE 6.6 “Center” (mean) of each Position label.

Position	Height	Weight
Defense	0.6446	0.6385
Forward	0.4742	0.4945
Reserve	0.3434	0.3615
Goal	0.1506	0.0538

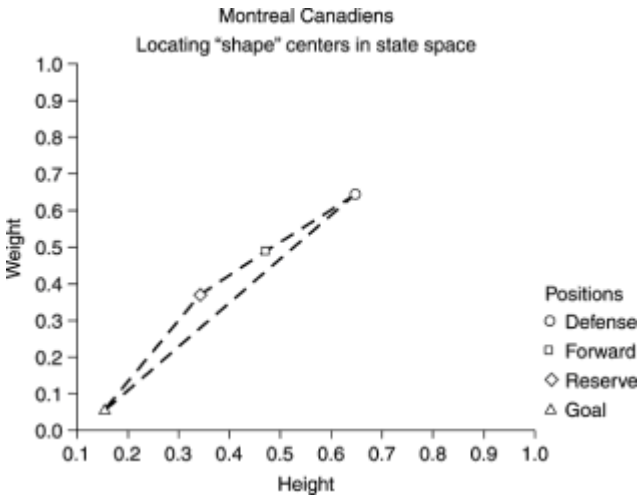

Figure 6.12 The “centers” (mean values on both dimensions) of each set of label values are located in state space. Joining the points makes a “shape.” Here the shape is nearly straight line.

In this case the labels do nearly fall on a straight line. As this is the case, numerating the

alpha labels can be imagined as starting with one end of the line as “0,” say with “Goal” near the zero point, and setting the other end of the line, at “Defense,” to “1.” The intervening values are set in proportion. From looking at the state space map, it seems about right to set the value of “Reserve” to about 0.4 and “Forward” to about 0.6. Usually, however, Shapes are not much like a straight line. Also, in higher dimensionalities, finding the appropriate ordering is more difficult.

6.3 Joint Distribution Tables

A different sort of problem arises if there is no numeric variable present. When there is at least one numeric variable present, it is used to set an order and spacing for the alpha variables. Without a numeric variable present, there is nothing to “calibrate” the alpha variables against. The problem is how to find any sort of logical ordering revealed by the relationships between the alpha values. The solution comes in steps. The first is to discover how the alpha values of one variable relate to the alpha values of another variable, or variables. A useful way to begin this discovery is by using a joint frequency, or joint distribution, table.

6.3.1 Two-Way Tables

A two-way table shows the joint frequency listing of alpha values between two variables. As an illustration we will return to the Montreal Canadiens. This time both height and weight will be turned into categorical values, and from these values a two-way joint frequency table is constructed. For ease of explanation, the heights are categorized as “tall,” “average,” and “short.” The weights are categorized as “heavy,” “medium,” and “light.” The categories are abbreviated “T,” “A,” “S,” and “H,” “M,” “L,” respectively.

The category boundaries are set to divide weight and height into three approximately equally sized categories as shown in Tables 6.7 and 6.8.

TABLE 6.7 Height category division criteria.

Height	Category
x6.22	T
x5.94 and x6.22	A
x5.94	S

TABLE 6.8 Weight category division criteria.

Weight	Category
x213.33	H
x191.66 and x213.33	M
x191.66	L

These three categories, or alpha labels, generate a two-way table with nine entries, one for each combination of labels. The categories for each player are shown in Table 6.9, and the cross-tabulation table is shown in Table 6.10. Figure 6.13 illustrates the distribution graphically.

TABLE 6.9 Players' names, actual height, normalized height, weights, and categories.

Name	Height	NmHt	Wt	CatHt	CatWt
Peter Popovic	6.5	1	235	T	H
Vladimir Malakhov	6.3	0.759036	227	T	H
Turner Stevenson	6.25	0.698795	220	T	H
Scott Thornton	6.25	0.698795	219	T	H
Jonas Hoglund	6.25	0.698795	215	T	H
Stephane Quintal	6.25	0.698795	215	T	H

Dave Manson	6.17	0.60241	219	A	H
Patrice Brisebois	6.17	0.60241	204	A	M
Craig Rivet	6.17	0.60241	195	A	M
Mick Vukota	6.08	0.493976	225	A	H
Zarley Zalapski	6.08	0.493976	215	A	H
Patrick Poulin	6.08	0.493976	210	A	M
Martin Rucinsky	6.08	0.493976	205	A	M
Igor Ulanov	6.08	0.493976	205	A	M
Marc Bureau	6.08	0.493976	202	A	M
Shayne Corson	6.08	0.493976	199	A	M
Vincent Damphousse	6.08	0.493976	191	A	L
Brian Savage	6.08	0.493976	191	A	L
Benoit Brunet	6	0.39759	194	A	M
Brett Clark	6	0.39759	182	A	L
Sebastien Bordeleau	5.92	0.301205	188	S	L
Jocelyn Thibault	5.92	0.301205	170	S	L
Eric Houde	5.83	0.192771	186	S	L
Mark Recchi	5.83	0.192771	185	S	L
Saku Koivu	5.83	0.192771	182	S	L
Andy Moog	5.67	0	177	S	L

TABLE 6.10 Cross-tabulation.

	H	M	L	Total
T	6	0	0	6
A	3	8	3	14
S	0	0	6	6
Total	9	8	9	26

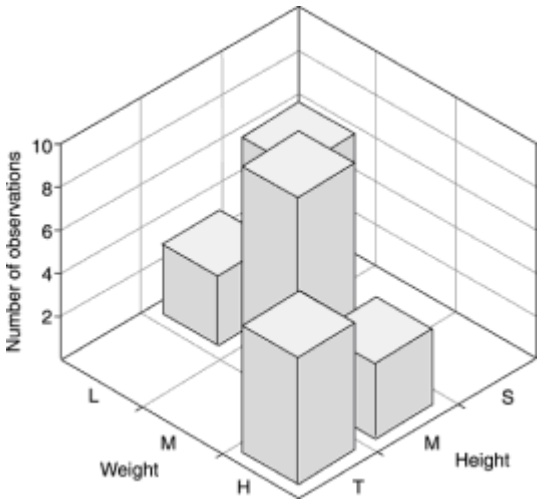

Figure 6.13 Bivariate histogram showing the joint distributions of the categories for weight and height of the Canadiens.

Notice that some of the categories overlap each other. It is these overlaps that allow an appropriate ordering for the categories to be discovered.

In this example, since the meaning of the labels is known, the ordering may appear intuitive. However, since the labels are arbitrary, and applied meaningfully only for ease in the example, they can be validly restated. Table 6.11 shows the same information as in Table 6.10, but with different labels, and reordered. Is it now intuitively easy to see what the ordering should be?

TABLE 6.11 Restated cross-tabulation.

	A	B	C	Total
X	3	3	8	14
Y	0	6	0	6
Z	6	0	0	6
Total	9	9	8	26

Table 6.11 contains exactly the same information as Table 6.10, but has made intuitive ordering difficult or impossible. It is possible to use this information to reconstruct an appropriate ordering, albeit not intuitively. For ease of understanding the previous labeling system is used, although the actual labels used, so long as consistently applied, are not important to recovering an ordering.

Restating the cross-tabulation of Table 6.10 in a different form shows how this recovery begins. Table 6.12 lists the number of players in each of the possible categories.

TABLE 6.12 Category/count tabulation.

Weight	Height	Count
H	T	6
H	A	3
H	S	0
M	T	0
M	A	8
M	S	0

L	T	0
L	A	3
L	S	6

The information in Table 6.12 represents a sort of jigsaw puzzle. Although in this example the categories in all of the tables are shown appropriately ordered to clarify explanation, the real situation is that the ordering is unknown and that needs to be discovered. What is known are the various frequencies for each of the category couplings, which are pairings here as there are only two variables. From these, the shape of the jigsaw pieces can be discovered.

Figure 6.14(a) shows the pieces that correspond to Weight = “H.” Altogether there are nine players with weight “H.” Six of them have height “T,” three of them have height “A,” and none of them have height “S.” Of the three possible pieces corresponding to H/T, H/A, and H/S, only the first two have any players in them. The figure shows the two pieces. Inside each box is a symbol indicating the label and how many players are accounted for. If the symbols are in brackets, it indicates that only part of the total number of players in the class are accounted for. Thus in the left-hand box, the top (6H) refers to six of the players with label “H,” and there remain other players with label “H” not accounted for. The lower 6T refers to all six players with height label “T.” The dotted lines at each end of the incomplete classes indicate that they need to be joined to other pieces containing members of the same class, that is, possessing the same label. The dotted lines are at each end because they could be joined together at either end. Similar pieces can be constructed for all of the label classes. These two example pieces can be joined together to form the piece shown in Figure 6.14(b).

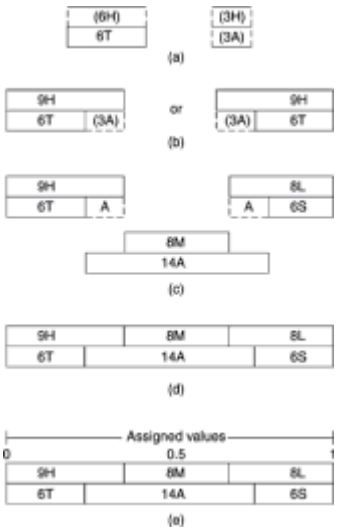

Figure 6.14 Shapes for all players with weight = “H” (a), two possible assembled shapes for the 9H/6T/3A categories (b), shapes created for each of the category combinations (c), fitting the pieces together recovers an appropriate ordering (d), and showing a straight-forward way of finding a numeration of each variable’s three segments (e).

Figure 6.14(b) shows the shape of the piece for all players with Weight = “H.” This is built from the two pieces in Figure 6.14(a). There are nine players with weight “H.” Of these, six have height “T” and three have height “A.” The appropriate jigsaw piece can be assembled in two ways; the overlapping “T” and “A” can be moved. Since the nine “H” (heavy) players cover all of the “T” (tall) players, the “H” and “T” parts are shown drawn solidly. The three “A” are left as part of some other pairing, and shown dotted. Similar shapes can be generated for the other category pairings. Figure 6.14(c) shows those.

For convenience, Figure 6.14(c) shows the pieces in position to fit together. In fact, the top and bottom sections can slide over each other to appropriate positions. Fitting them together so that the matching pieces adjoin can only be completed in two ways. Both are identical except that in one “H” and “T” are on the left, with “S” and “L” on the right. The other configuration is a mirror image.

Fitting the pieces together reveals the appropriate *order* for the values to be placed in relation to each other. This is shown in Figure 6.14(d). Which end corresponds to “0” and which to “1” on a normalized scale is not possible to determine. Since in the example there are only three values in each variable, numerating them is straightforward. The values are assigned in the normalized range of 0–1, and values are assigned as shown in Figure 6.14(e).

Having made an arbitrary decision to assign the value 0 to “H” and “T,” the actual numerical relationship in this example is now inverted. This means that larger values of weight and height are estimated as lower normalized values. The *relationship* remains intact but the numbers go in the “wrong” direction. Does this matter? Not really. For modeling purposes it is finding and keeping appropriate relationships that is paramount. If it ever becomes possible to anchor the estimated values to the real world, the accuracy of the predictions of real-world values is unaffected by the direction of increase in the estimates. If the real-world values remain unknown, then, when numeric predictions are made by the final model, they will be converted back into their appropriate alpha value, which is internally consistent within the model. The alpha value predictions will be unaffected by the internal numerical representation used by the model.

Although very simplified, how well does this numeration of the alpha values work? For convenience Table 6.13 shows the normalized weights and normalized heights with the estimated valves uninverted. This makes comparison easier.

TABLE 6.13 Comparison of recovered values with normalized values.

Normalized height	Estimated height	Normalized weight	Estimated weight
1	1	1	1
0.759036	1	0.876923	1
0.698795	1	0.769231	1
0.698795	1	0.753846	1
0.698795	1	0.692308	1
0.698795	1	0.692308	1
0.60241	0.5	0.753846	1
0.60241	0.5	0.523077	0.5
0.60241	0.5	0.384615	0.5
0.493976	0.5	0.846154	1
0.493976	0.5	0.692308	1
0.493976	0.5	0.615385	0.5
0.493976	0.5	0.538462	0.5
0.493976	0.5	0.538462	0.5
0.493976	0.5	0.492308	0.5
0.493976	0.5	0.446154	0.5
0.493976	0.5	0.323077	0

0.493976	0.5	0.323077	0
0.39759	0.5	0.369231	0.5
0.39759	0.5	0.184615	0
0.301205	0	0.276923	0
0.301205	0	0	0
0.192771	0	0.246154	0
0.192771	0	0.230769	0
0.192771	0	0.184615	0
0	0	0.107692	0

6.3.2 More Values, More Variables, and Meaning of the Numeration

The Montreal Canadiens example is very highly simplified. It has a very small number of instance values and only three alpha values in each variable. In any practically modelable data set, there are always far more instances of data available and usually far more variables and alpha labels to be considered. The numeration process continues using exactly the same principles as just described. With more data and more variables, the increased interaction between the variables allows finer discrimination of values to be made.

What has using this method achieved? Only discovering an appropriate order in which to place the alpha values. While the ordering is very important, the appropriate distance between the values has not yet been discovered. In other words, we can, from the example, determine the appropriate order for the labels of height and weight. We cannot yet determine if the difference between, say, “H” and “M” is greater or less than the difference between “M” and “L.” This is true in spite of the fact that “H” is assigned a value of 1, “M” of 0.5, and “L” of 0. At this juncture, no more can be inferred from the assignment H = 1, M = 0.5, L = 0 than could be inferred from H = 1, M = 0.99, L = 0, or H = 1, M = 0.01, L = 0.

Something can be inferred about the values between variables. Namely, when normalized values are being used, both “H” and “T” should have about the same value, and “M” and “A” should have about the same value, as should “L” and “S.” This does not suggest that

they share similar values in the real world, only that a consistent internal representation requires maintenance of the pattern of the relationship between them.

Even though the alpha labels are numerically ordered, it is only the ordering that has significance, not the value itself. It is sometimes possible to recover information about the appropriate separation of values in entirely alpha data sets. However, this is not always the case, as it is entirely possible that there is no meaningful separation between values. That is the inherent nature of alpha values. Steps toward recovering appropriate separation of values in entirely alpha data sets, if indeed such meaningful separation exists, are discussed in the next chapter dealing with normalizing and redistributing variables.

6.3.3 Dealing with Low-Frequency Alpha Labels and Other Problems

The joint frequency method of finding appropriate numerical labels for alpha values can only succeed when there is a sufficient and rich overlap of joint distributions. This is not always the case for all variables in all data sets. In any real-world data set, there is always enough richness of interaction among some of the variables that it is possible to numerate them using the joint frequency table approach. However, it is by no means always the case that the joint frequency distribution table is well enough populated to allow this method to work for all variables. In a very large data set, some of the cells, similar to those illustrated in Figure 6.13, are simply empty. How then to find a suitable numerical representation for those variables?

The answer lies in the fact that it is always possible to numerate some of the variables using this method. When such variables have been numerated, then they can be put into a numerical form of representation. With such a representation available in the data set, it becomes possible to numerate the remaining variables using the method discussed in the previous section dealing with state space. The alpha variables amenable to numeration using the joint frequency table approach are numerated. Then, constructing the manifold in state space using the numerated variables, values for the remaining variable instance values can be found.

6.4 Dimensionality

The preceding two parts of this chapter discussed finding an appropriate numerical representation for an alpha label value. In most cases, the discovered numeric representation, as so far discussed, is as a location on a manifold in state or phase space. This representation of the value has to be described as a position in phase space, which takes as many numbers as there are dimensions. In a 200-dimensional space, it would take a string of 200 numbers to indicate the value “gender = F,” and another similar string, with different values, to indicate “gender = M.” While this is a valid representation of the alpha values, it is hopelessly impractical and totally intractable to model. Adding 200

additional dimensions to the model simply to represent gender is impossible to deal with practically. The number of dimensions for alpha representation has to be reduced, and the method used is based on the principles of multidimensional scaling.

This explanation will use a metaphor different from that of a manifold for the points in phase space. Instead of using density to conjure up the image of a surface, each point will be regarded as being at the “corner” of a shape. Each line that can be drawn from point to point is regarded as an “edge” of a figure existing in space. An example is a triangle. The position of three points in space can be joined with lines, and the three points define the shape, size, and properties of the triangle.

6.4.1 Multidimensional Scaling

MDS is used specifically to “project” high-dimensionality objects into a lower-dimensional space, losing as little information as possible in the process. The key idea is that there is some inherent dimensionality of a representation. While the representation is made in more dimensions than is needed, not much information is lost. Forcing the representation into less dimensions than are “natural” for the representation does cause significant loss, producing “stress.” MDS aims at minimizing this stress, while also minimizing the number of dimensions the representation needs. As an example of how this is done, we will attempt to represent a triangle in one dimension—and see what happens.

6.4.2 Squashing a Triangle

A triangle is inherently a 2D object. It can be defined by three points in a state or phase space. All of the triangular points lie in a plane, which is a 2D surface. When represented in three dimensions, such as when printed on the page of this book, the triangle has some minute thickness. However, for practical purposes we ignore the thickness that is actually present and pretend that the triangle is really 2D. That is to say, mentally we can project the 3D representation of a triangle into two dimensions with very little loss of information. We do lose information about the actual triangle, say the thickness of the ink, since there is no thickness in two dimensions. Also lost is information about the actual flatness, or roughness, of the surface of the paper.

Since paper cannot be exactly flat in the real world, the printed lines of the triangle are minutely longer than they would be if the paper were exactly flat. To span the miniature hills and valleys on the paper’s surface, the line deviates ever so minutely from the shortest path between the two points. This may add, say, one-thousandth of one percent to the entire length of the line. This one-thousandth of one percent change in length of the line when the triangle is projected into 2D space is a measure of the stress, or loss of information, that occurs in projecting a triangle from three to two dimensions. But what happens if we try to project a triangle into one dimension? Can it even be done?

Figure 6.15 shows, in part, two right-angled triangles that are identical except for their