Outliers

The more data we are dealing with, the greater the chance of encountering erroneous values resulting from measurement error, data-entry error, or the like. If the erroneous value is in the same range as the rest of the data, it may be harmless. If it is well outside the range of the rest of the data (a misplaced decimal, for example), it may have a substantial effect on some of the data mining procedures we plan to use.

Values that lie far away from the bulk of the data are called *outliers*. The term *far away* is deliberately left vague because what is or is not called an outlier is an arbitrary decision. Analysts use rules of thumb such as "anything over three standard deviations away from the mean is an outlier," but no statistical rule can tell us whether such an outlier is the result of an error. In this statistical sense, an outlier is not necessarily an invalid data point, it is just a distant one.

The purpose of identifying outliers is usually to call attention to values that need further review. We might come up with an explanation looking at the data—in the case of a misplaced decimal, this is likely. We might have no explanation, but know that the value is wrong—a temperature of 178°F for a sick person. Or, we might conclude that the value is within the realm of possibility and leave it alone. All these are judgments best made by someone with *domain knowledge*, knowledge of the particular application being considered: direct mail, mortgage finance, and so on, as opposed to technical knowledge of statistical or data mining procedures. Statistical procedures can do little beyond identifying the record as something that needs review.

If manual review is feasible, some outliers may be identified and corrected. In any case, if the number of records with outliers is very small, they might be treated as missing data. How do we inspect for outliers? One technique is to sort the records by the first column (e.g., using the R function *order()*, then review the data for very large or very small values in that column. Then repeat for each successive column. Another option is to examine the minimum and maximum values of each column using R's *min()* and *max()* functions. For a more automated approach that considers each record as a unit, rather than each column in isolation, clustering techniques (see Chapter 14) could be used to identify clusters of one or a few records that are distant from others. Those records could then be examined.

Missing Values

Typically, some records will contain missing values. If the number of records with missing values is small, those records might be omitted. However, if we have a large number of variables, even a small proportion of missing values can affect a lot of records. Even with only 30 variables, if only 5% of the values are missing (spread randomly and independently among cases and variables), almost 80% of the records would have to be omitted from the analysis. (The chance that a given record would escape having a missing value is $0.95^{30} = 0.215$.)

An alternative to omitting records with missing values is to replace the missing value with an imputed value, based on the other values for that variable across all records. For example, if among 30 variables, household income is missing for a particular record, we might substitute the mean household income across all records. Doing so does not, of course, add any information about how household income affects the outcome variable. It merely allows us to proceed with the analysis and not lose the information contained in this record for the other 29 variables. Note that using such a technique will understate the variability in a dataset. However, we can assess variability and the performance of our data mining technique using the validation data, and therefore this need not present a major problem. One option is to replace missing values using fairly simple substitutes (e.g., mean, median). More sophisticated procedures do exist—for example, using linear regression, based on other variables, to fill in the missing values. These methods have been elaborated mainly for analysis of medical and scientific studies, where each patient or subject record comes at great expense. In data mining, where data are typically plentiful, simpler methods usually suffice. Table 2.7 shows some R code to illustrate the use of the median to replace missing values. Since the data are complete to begin with, the first step is an artificial one of creating some missing records for illustration purposes. The median is used for imputation, rather than the mean, to preserve the integer nature of the counts for bedrooms.

Table 2.7 Missing Data

code for imputing missing data with median

```
# To illustrate missing data procedures, we first convert a few entries for
```

```
# To illustrate missing data procedures, we first convert a few entries for
# bedrooms to NA's. Then we impute these missing values using the median of the
# remaining values.
rows.to.missing <- sample(row.names(housing.df), 10)
housing.df[rows.to.missing,]$BEDROOMS <- NA
summary(housing.df$BEDROOMS)  # Now we have 10 NA's and the median of the
# remaining values is 3.

# replace the missing values using the median of the remaining values.
# use median() with na.rm = TRUE to ignore missing values when computing the median.
housing.df[rows.to.missing,]$BEDROOMS <- median(housing.df$BEDROOMS, na.rm = TRUE)

summary(housing.df$BEDROOMS)

Partial Output
```

```
> housing.df[rows.to.missing,]$BEDROOMS <- NA
> summary(housing.df$BEDROOMS)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
   1.00    3.00    3.00    3.23    4.00    9.00      10

> housing.df[rows.to.missing,]$BEDROOMS <- median(housing.df$BEDROOMS, na.rm = TRUE)
> summary(housing.df$BEDROOMS)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    3.00    3.00    3.23    4.00    9.00
```

Some datasets contain variables that have a very large number of missing values. In other words, a measurement is missing for a large number of records. In that case, dropping records with missing values will lead to a large loss of data. Imputing the missing values might also be useless, as the imputations are based on a small number of existing records. An alternative is to examine the importance of the predictor. If it is not very crucial, it can be dropped. If it is important, perhaps a proxy variable with fewer missing values can be used instead. When such a predictor is deemed central, the best solution is to invest in obtaining the missing data.

Significant time may be required to deal with missing data, as not all situations are susceptible to automated solutions. In a messy dataset, for example, a "0" might mean two things: (1) the value is missing, or (2) the value is actually zero. In the credit industry, a "0" in the "past due" variable might mean a customer who is fully paid up, or a customer with no credit history at all—two very different situations. Human judgment may be required for individual cases or to determine a special rule to deal with the situation.

Normalizing (Standardizing) and Rescaling Data

Some algorithms require that the data be normalized before the algorithm can be implemented effectively. To normalize a variable, we subtract the mean from each value and then divide by the standard deviation. This operation is also sometimes called *standardizing*. In R, function *scale()* performs this operation. In effect, we are expressing each value as the "number of standard deviations away from the mean," also called a *z-score*.

Normalizing is one way to bring all variables to the same scale. Another popular approach is rescaling each variable to a [0,1] scale. This is done by subtracting the minimum value and then dividing by the range. Subtracting the minimum shifts the variable origin to zero. Dividing by the range shrinks or expands the data to the range [0,1]. In R, rescaling can be done using function *rescale()* in the scales package.

To consider why normalizing or scaling to [0,1] might be necessary, consider the case of clustering. Clustering typically involves calculating a distance measure that reflects how far each record is from a cluster center or from other records. With multiple variables, different units will be used: days, dollars, counts, and so on. If the dollars are in the thousands and everything else is in the tens, the dollar variable will come to dominate the distance measure. Moreover, changing units from, say, days to hours or months, could alter the outcome completely.

Data mining software typically have an option to normalize the data in those algorithms where it may be required. It is an option rather than an automatic feature of such algorithms, because there are situations where we want each variable to contribute to the distance measure in proportion to its original scale.

2.5 Predictive Power and Overfitting

In supervised learning, a key question presents itself: How well will our prediction or classification model perform

when we apply it to new data? We are particularly interested in comparing the performance of various models so that we can choose the one we think will do the best when it is implemented in practice. A key concept is to make sure that our chosen model generalizes beyond the dataset that we have at hand. To assure generalization, we use the concept of *data partitioning* and try to avoid *overfitting*. These two important concepts are described next.

Overfitting

The more variables we include in a model, the greater the risk of overfitting the particular data used for modeling. What is overfitting?

In Table 2.8, we show hypothetical data about advertising expenditures in one time period and sales in a subsequent time period. A scatter plot of the data is shown in Figure 2.2. We could connect up these points with a smooth but complicated function, one that interpolates all these data points perfectly and leaves no error (residuals). This can be seen in Figure 2.3. However, we can see that such a curve is unlikely to be accurate, or even useful, in predicting future sales on the basis of advertising expenditures. For instance, it is hard to believe that increasing expenditures from $400 to $500 will actually decrease revenue.

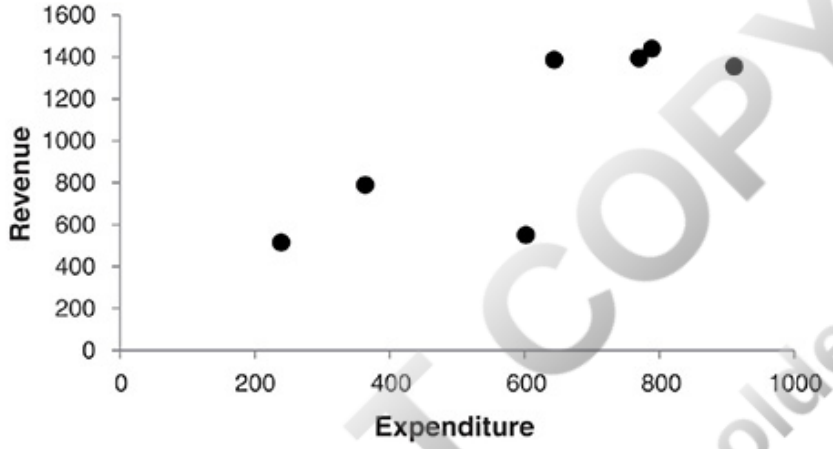

Figure 2.2 scatter plot for advertising and sales data

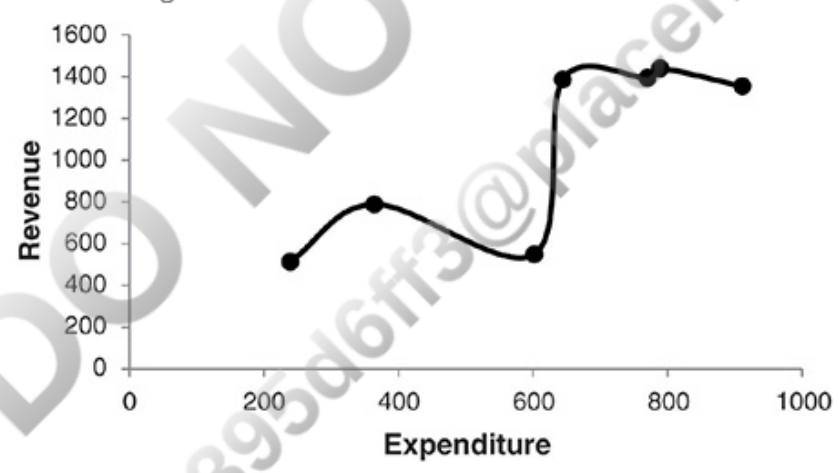

Figure 2.3 Overfitting: This function fits the data with no error

Table 2.8

Advertising	Sales
239	514
364	789
602	550
644	1386
770	1394
789	1440

A basic purpose of building a model is to represent relationships among variables in such a way that this representation will do a good job of predicting future outcome values on the basis of future predictor values. Of course, we want the model to do a good job of describing the data we have, but we are more interested in its performance with future data.

In the hypothetical advertising example, a simple straight line might do a better job than the complex function in terms of predicting future sales on the basis of advertising. Instead, we devised a complex function that fit the data perfectly, and in doing so, we overreached. We ended up modeling some variation in the data that is nothing more than chance variation. We mistreated the noise in the data as if it were a signal.

Similarly, we can add predictors to a model to sharpen its performance with the data at hand. Consider a database of 100 individuals, half of whom have contributed to a charitable cause. Information about income, family size, and zip code might do a fair job of predicting whether or not someone is a contributor. If we keep adding additional predictors, we can improve the performance of the model with the data at hand and reduce the misclassification error to a negligible level. However, this low error rate is misleading, because it probably includes spurious effects, which are specific to the 100 individuals, but not beyond that sample.

For example, one of the variables might be height. We have no basis in theory to suppose that tall people might contribute more or less to charity, but if there are several tall people in our sample and they just happened to contribute heavily to charity, our model might include a term for height—the taller you are, the more you will contribute. Of course, when the model is applied to additional data, it is likely that this will not turn out to be a good predictor.

If the dataset is not much larger than the number of predictor variables, it is very likely that a spurious relationship like this will creep into the model. Continuing with our charity example, with a small sample just a few of whom are tall, whatever the contribution level of tall people may be, the algorithm is tempted to attribute it to their being tall. If the dataset is very large relative to the number of predictors, this is less likely to occur. In such a case, each predictor must help predict the outcome for a large number of cases, so the job it does is much less dependent on just a few cases, which might be flukes.

Somewhat surprisingly, even if we know for a fact that a higher-degree curve is the appropriate model, if the model-fitting dataset is not large enough, a lower-degree function (that is not as likely to fit the noise) is likely to perform better in terms of predicting new values. Overfitting can also result from the application of many different models, from which the best performing model is selected.

Creation and Use of Data Partitions

At first glance, we might think it best to choose the model that did the best job of classifying or predicting the outcome variable of interest with the data at hand. However, when we use the same data both to develop the model and to assess its performance, we introduce an “optimism” bias. This is because when we choose the model that works best with the data, this model’s superior performance comes from two sources:

- A superior model
- Chance aspects of the data that happen to match the chosen model better than they match other models

The latter is a particularly serious problem with techniques (such as trees and neural nets) that do not impose linear or other structure on the data, and thus end up overfitting it.

To address the overfitting problem, we simply divide (partition) our data and develop our model using only one of the partitions. After we have a model, we try it out on another partition and see how it performs, which we can measure in several ways. In a classification model, we can count the proportion of held-back records that were misclassified. In a prediction model, we can measure the residuals (prediction errors) between the predicted values and the actual values. This evaluation approach in effect mimics the deployment scenario, where our model is applied to data that it hasn’t “seen.”

We typically deal with two or three partitions: a training set, a validation set, and sometimes an additional test set. Partitioning the data into training,