Oversampling Rare Events in Classification Tasks

If the event we are interested in classifying is rare, for example, customers purchasing a product in response to a mailing, or fraudulent credit card transactions, sampling a random subset of records may yield so few events (e.g., purchases) that we have little information on them. We would end up with lots of data on nonpurchasers and non-fraudulent transactions but little on which to base a model that distinguishes purchasers from nonpurchasers or fraudulent from non-fraudulent. In such cases, we would want our sampling procedure to overweight the rare class (purchasers or frauds) relative to the majority class (nonpurchasers, non-frauds) so that our sample would end up with a healthy complement of purchasers or frauds.

Assuring an adequate number of responder or "success" cases to train the model is just part of the picture. A more important factor is the costs of misclassification. Whenever the response rate is extremely low, we are likely to attach more importance to identifying a responder than to identifying a non-responder. In direct-response advertising (whether by traditional mail, e-mail, or web advertising), we may encounter only one or two responders for every hundred records—the value of finding such a customer far outweighs the costs of reaching him or her. In trying to identify fraudulent transactions, or customers unlikely to repay debt, the costs of failing to find the fraud or the nonpaying customer are likely to exceed the cost of more detailed review of a legitimate transaction or customer.

If the costs of failing to locate responders are comparable to the costs of misidentifying responders as non-responders, our models would usually achieve highest overall accuracy if they identified everyone as a non-responder (or almost everyone, if it is easy to identify a few responders without catching many nonresponders). In such a case, the misclassification rate is very low—equal to the rate of responders—but the model is of no value.

More generally, we want to train our model with the asymmetric costs in mind so that the algorithm will catch the more valuable responders, probably at the cost of "catching" and misclassifying more non-responders as responders than would be the case if we assume equal costs. This subject is discussed in detail in Chapter 5.

Preprocessing and Cleaning the Data

Types of Variables

There are several ways of classifying variables. Variables can be numerical or text (character/string). They can be continuous (able to assume any real numerical value, usually in a given range), integer (taking only integer values), categorical (assuming one of a limited number of values), or date. Categorical variables can be either coded as numerical (1, 2, 3) or text (payments current, payments not current, bankrupt). Categorical variables can be unordered (called *nominal variables*) with categories such as North America, Europe, and Asia; or they can be ordered (called *ordinal variables*) with categories such as high value, low value, and nil value.

Continuous variables can be handled by most data mining routines with the exception of the naive Bayes classifier, which deals exclusively with categorical predictor variables. The machine learning roots of data mining grew out of problems with categorical outcomes; the roots of statistics lie in the analysis of continuous variables. Sometimes, it is desirable to convert continuous variables to categorical variables. This is done most typically in the case of outcome variables, where the numerical variable is mapped to a decision (e.g., credit scores above a certain threshold mean "grant credit," a medical test result above a certain threshold means "start treatment").

For the West Roxbury data, Table 2.5 presents some R statements to review the variables and determine what type (class) R thinks they are, and to determine the number of levels in a categorical variable (= factor variable in R).

Table 2.5 Reviewing variables in R

code for reviewing variables

```
names(housing.df)  # print a list of variables to the screen.
t(t(names(housing.df)))  # print the list in a useful column format
colnames(housing.df)[1] <- c("TOTAL_VALUE")  # change the first column's name
class(housing.df$REMODEL) # REMODEL is a factor variable
class(housing.df[ ,14]) # Same.
levels(housing.df[, 14])  # It can take one of three levels
class(housing.df$BEDROOMS)  # BEDROOMS is an integer variable
class(housing.df[, 1])  # Total Value is a numeric variable
```

```
class(housing.df[, 1])  # Total_Value is a numeric variable
```

Partial Output

```
> t(t(names(housing.df)))
      [,1]
 [1,] "TOTAL_VALUE"
 [2,] "TAX"
 [3,] "LOT.SQFT"
 [4,] "YR.BUILT"
 [5,] "GROSS.AREA"
 [6,] "LIVING.AREA"
 [7,] "FLOORS"
 [8,] "ROOMS"
 [9,] "BEDROOMS"
[10,] "FULL.BATH"
[11,] "HALF.BATH"
[12,] "KITCHEN"
[13,] "FIREPLACE"
[14,] "REMODEL"

> class(housing.df$REMODEL)
[1] "factor"

> levels(housing.df[, 14])
[1] "None"   "Old"    "Recent"
```

Handling Categorical Variables

Categorical variables can also be handled by most data mining routines, but often require special handling. If the categorical variable is ordered (age group, degree of creditworthiness, etc.), we can sometimes code the categories numerically (1, 2, 3, ...) and treat the variable as if it were a continuous variable. The smaller the number of categories, and the less they represent equal increments of value, the more problematic this approach becomes, but it often works well enough.

Nominal categorical variables, however, often cannot be used as is. In many cases, they must be decomposed into a series of binary variables, called *dummy variables*. For example, a single categorical variable that can have possible values of "student," "unemployed," "employed," or "retired" would be split into four separate dummy variables:

Student—Yes/No

Unemployed—Yes/No

Employed—Yes/No

Retired—Yes/No

In many cases, only three of the dummy variables need to be used; if the values of three are known, the fourth is also known. For example, given that these four values are the only possible ones, we can know that if a person is neither student, unemployed, nor employed, he or she must be retired. In some routines (e.g., linear regression and logistic regression), you should not use all four variables—the redundant information will cause the algorithm to fail. Note, also, that typical methods of creating dummy variables will leave the original categorical variable intact; obviously you should not use both the original variable and the dummies. The R code to create binary dummies from a categorical (factor) variable is given in Table 2.6.

Table 2.6 Creating Dummy Variables in R

code for creating binary dummies (indicators)

```
# use model.matrix() to convert all categorical variables in the data frame into
# a set of dummy variables. We must then turn the resulting data matrix back into
# a data frame for further work.
xtotal <- model.matrix(~ 0 + BEDROOMS + REMODEL, data = housing.df)
```

```
xtotal$BEDROOMS[1:5]  # will not work because xtotal is a matrix
xtotal <- as.data.frame(xtotal)
t(t(names(xtotal)))  # check the names of the dummy variables
head(xtotal)
xtotal <- xtotal[, -4]  # drop one of the dummy variables.
# In this case, drop REMODELRecent.
```

Partial Output

```
> t(t(names(xtotal)))  # Check the names of the dummy variables.
     [,1]
[1,] "BEDROOMS"
[2,] "REMODELNone"
[3,] "REMODELOld"
[4,] "REMODELRecent"

> head(xtotal)
  BEDROOMS REMODELNone REMODELOld REMODELRecent
1        3           1          0             0
2        4           0          0             1
3        4           1          0             0
4        5           1          0             0
5        3           1          0             0
6        3           0          1             0
```

Variable Selection

More is not necessarily better when it comes to selecting variables for a model. Other things being equal, parsimony, or compactness, is a desirable feature in a model. For one thing, the more variables we include and the more complex the model, the greater the number of records we will need to assess relationships among the variables. Fifteen records may suffice to give us a rough idea of the relationship between Y and a single predictor variable X. If we now want information about the relationship between Y and 15 predictor variables $X_1, \ldots, X_{15}$, 15 records will not be enough (each estimated relationship would have an average of only one record's worth of information, making the estimate very unreliable). In addition, models based on many variables are often less robust, as they require the collection of more variables in the future, are subject to more data quality and availability issues, and require more data cleaning and preprocessing.

How Many Variables and How Much Data?

Statisticians give us procedures to learn with some precision how many records we would need to achieve a given degree of reliability with a given dataset and a given model. These are called "power calculations" and are intended to assure that an average population effect will be estimated with sufficient precision from a sample. Data miners' needs are usually different, because the focus is not on identifying an average effect but rather on predicting individual records. This purpose typically requires larger samples than those used for statistical inference. A good rule of thumb is to have 10 records for every predictor variable. Another rule, used by Delmaster and Hancock (2001, p. 68) for classification procedures, is to have at least $6 \times m \times p$ records, where m is the number of outcome classes and p is the number of variables.

In general, compactness or parsimony is a desirable feature in a data mining model. Even when we start with a small number of variables, we often end up with many more after creating new variables (such as converting a categorical variable into a set of dummy variables). Data visualization and dimension reduction methods help reduce the number of variables so that redundancies and information overlap are reduced.

Even when we have an ample supply of data, there are good reasons to pay close attention to the variables that are included in a model. Someone with domain knowledge (i.e., knowledge of the business process and the data) should be consulted, as knowledge of what the variables represent is typically critical for building a good model and avoiding errors. For example, suppose we're trying to predict the total purchase amount spent by customers, and we have a few predictor columns that are coded X_1, X_2, X_3, …, where we don't know what those codes mean. We might find that X_1 is an excellent predictor of the total amount spent. However, if we discover that X_1 is the amount spent on shipping, calculated as a percentage of the purchase amount, then obviously a model that uses shipping amount cannot be used to predict purchase amount, because the shipping amount is not known until the transaction is completed. Another example is if we are trying to predict loan default at the time a customer applies for a loan. If our

completed. Another example is if we are trying to predict loan default at the time a customer applies for a loan. If our dataset includes only information on approved loan applications, we will not have information about what distinguishes defaulters from non-defaulters among denied applicants. A model based on approved loans alone can therefore not be used to predict defaulting behavior at the time of loan application, but rather only once a loan is approved.

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 $0.95^{30} = 0.215$.)