5

PREPARING TRAINING AND TEST DATASETS

THE LOGIC OF CROSS-VALIDATION

Earlier, we discussed how cross-validation (CV) works as a sort of quality control mechanism in data mining, and pointed out how CV methods contrast in an interesting manner with conventional tests for statistical significance. We will now discuss explicitly the logic of CV, and then provide a guide for how one carries out this technique in practice using a number of statistical packages.

Many data mining texts deal with the logic of CV in a rather cursory fashion. The focus is on its practical application: how CV presents a solution to one or another problem that tends to be encountered when employing computationally intensive methods with big data. In some texts CV is presented as a way to prevent overfitting (Nisbet, Elder, and Miner, 2009; Kuhn and Johnson 2013); in others, as a means for model selection (Murphy 2012); and in others, as a way to evaluate model accuracy (Han, Kamber, and Pei 2012). In fact, CV is all of these, but the reasons why this is the case, and indeed why these problems are interlinked, tend to be left rather obscure. We here attempt to fill in this conceptual gap.

The problems data miners address with CV methods are ones which are central and familiar in scientific practice. In the scientific investigation of any particular problem, the result found in a single study can never be taken too seriously, by itself, as a valid description of how the world generally works. This is because the chance nature of sampling, and the possibility of random occurrence in an experiment, render it all too possible that findings from a single study are the result of the confluence of particular fortuitous circumstances. For results to be believed and accepted, they must obtain support from multiple subsequent investigations. In short, results must be replicated *independently,* ideally by entirely different researchers.

CV procedures permit researchers to employ this logic within a single investigation. In the simplest method of CV, researchers randomly split their data into subsamples prior to building a predictive model. Because the data splitting is *random,* for the limited purposes of the investigation at hand the subsamples generated constitute independent sets of observations. They are not independent in a global sense, since they were drawn from the same population (i.e. the full data set); but within the universe defined for the study at hand, and defined by the data that we are using, the subgroups are rendered independent of each other through randomization. Researchers build a model using one set of observations and then test it on another. This latter step represents an *independent* test of the model's accuracy.

Randomization is typically used to ensure that, on average, the subgroups produced are as similar as possible in terms of relevant characteristics (see e.g. Rubin 1978). However, it has another convenient result, which data mining CV methods exploit: the random variation among subgroups produced through random assignment. The idiosyncratic features of any particular randomly created subgroup are by definition unlikely to recur in the other subgroups. In contrast, empirical regularities across the subgroups are likely to be characteristics of the population as a whole—that is, they will tell us the *signal* to which we want to attend; the random variation across the subsamples can be thought of as the *noise* which we would like to analytically separate out.

This is of particular importance because of the very flexibility and power of data mining methods for creating predictive models. Because models such as neural networks and partition trees are capable of fitting themselves tightly to the data, they are highly susceptible to permitting the noise to play a greater role than one would wish in generating the model built on any particular set of observations. The very plasticity and power of the models becomes, in this sense, a curse. They will produce a model which is highly accurate, attaining near-perfection in predictive accuracy for the particular set of observations upon which it was built. But this apparent result is deceptive, or "overoptimistic" in the words of an early researcher (Larson 1931), for such a model will not perform well at all out of sample. This is what data miners refer to as *overfitting.* Using CV for model selection and evaluation can help reduce the possibility of its occurrence.

Consider why this is ultimately important. Data mining algorithms can be powerful tools for prediction, and can thus improve practical diagnostic ability. That is, if a data mining algorithm is trained on a set of data in which researchers have access to the true value of the outcome variable, and validated on an independent set where values of the outcome are also known, it can later be deployed in data in which the outcome variable, the quantity of interest, is *not* known. For example, data mining tools can potentially improve practitioners' ability to distinguish cancerous from noncancerous cells. Given this practical application, it is all the more important that the models be accurate; and this is ensured through rigorous application of CV for model selection and evaluation.

We have one final comment on the importance of CV methods. It has become abundantly clear to us in the course of our research that the predictive strength of computationally intensive data mining methods comes at a cost, in terms of the ability of the models to be fully comprehended by humans. Kuhn and Johnson (2013) describe this as the "tension between prediction and interpretation." Data mining models often garner their power through increasing complexity, and this renders them rather dizzying if not simply opaque to human analysts. But Kuhn and Johnson argue that, especially in life-and-death situations, this ought not to matter, and that indeed preferring a comprehensible but relatively poor-performing model to one which is a "black box" but is highly predictive would be "unethical." In the context of models whose results cannot be readily understood, but which have the appearance of being highly accurate, CV appears as an essential means of generating trust through rigorous testing.

In sum, CV provides an independent test of the model developed with a data mining technique. It assists with both choosing the "best" model (model selection) in terms of its capacity for out-of-sample prediction, and with evaluating the "true" predictive capacity of a given model (model evaluation). This helps guard against the possibility of selecting a model which is too dependent on the particular data it was built on—that is, it guards against overfitting. All of these functions are connected by a common reference to the logic of the independent test and the trustworthiness of replicable results. Now we present a brief discussion of different methods of CV, and then proceed to show how CV is performed using a number of statistical programs.

CROSS-VALIDATION METHODS: AN OVERVIEW

"Independent" datasets can be generated in a number of ways. The simplest conceptually, but the most complicated in terms of the actual work required, is to work with separately collected data. If we have built a predictive model for mortality using data from one hospital, we can test it on data gathered at a different hospital. However, this situation, though perhaps desirable, is fairly rare. Data collection is expensive, and it is unlikely that researchers and funders would want to double research costs simply so that predictive modelers can have a clean test dataset.

Otherwise, there are three methods for generating independent data from a single dataset: *bootstrapping*, *random holdback*, and *k-fold*. The first involves randomly sampling, with replacement, from the data that we have. Typically this is done a large number of times, creating many separate datasets of the same size as our original dataset. If we believe that our original data were a relatively random sample drawn from a population, then bootstrapping from it provides an unbiased method of generating a random sample of all possible random samples. Each of the multiple bootstrapped samples can be used to analyze the data, giving a range of results (coefficients or predicted probabilities, for example) which can then be averaged to obtain an overall result.

Bootstrapping presents certain advantages over other methods that will be discussed below. In particular, through bootstrapping we can generate not only an estimate of the error rate but also a standard error of the prediction, as well as a standard error of the error rate (Efron 1979, 1983; Efron and Gong 1983).[1] Moreover, bootstrapping produces a *nonparametric* standard error—which does not depend upon distributional assumptions that may not hold empirically—and indeed has entered into conventional statistical modeling primarily in this capacity. Finally, because bootstrapping generates numerous datasets, rather than just one, on which to test the model, it results in a "smoothed" estimate of the error rate. However, bootstrapping is quite computationally intensive and can be time-consuming to use with large datasets.

The bootstrapping approach is particularly valuable when one has a small sample to begin with, where statistical power is really an issue. In other cases, however, we have data to burn—increasingly true in the era of big data. In this latter situation, we can adopt a much simpler approach than bootstrapping, called *random holdback*: we can simply randomly split our data into training and test sets, build our model using the first, and test it on the second. We can decide to split the data between the training and test parts in whatever proportions we want—50/50, 70/30, and so on.

Alternatively, we can take a third approach, using what is called *k*-fold cross-validation. This involves dividing the data up randomly into *k* parts (or folds) of equal size, where *k* is a number chosen by the researcher. (Typical values for *k*, used in popular software packages, are 5 or 10 folds.) The researcher then builds *k* separate models, each using all but one of the folds and then tested on this remaining one. In essence, then, we have *k* training and *k* test sets, and each of the resulting *k* models is tested on data which were not used in its generation. Results can then be combined by averaging, or we can choose the best-fitting model. This method is also good for relatively small datasets.

We'll now demonstrate how to perform *k*-fold and holdback CV using several different statistical software packages.

STATA

Stata is not really a data mining package, so CV isn't built into it in such a manner as to make it particularly easy to use. Holdback CV needs to be done "by hand," as far as we know.

In order to perform holdback CV, the first thing that needs to be done is to randomly divide the data into two parts. Stata permits random sampling, which will allow the researcher to create separate datasets, but we think it is far easier to simply generate a variable that permits random division. One can do this by generating a random binomial with the code:

```
gen x = rbinomial(n, p)
```

TABLE 5.1 Results from Stata's Crossfold Program

Crossfold number	*Root mean square error*
1	0.3352256
2	0.3308182
3	0.3365854
4	0.3280907
5	0.3264875

This call generates a new variable named *x* which is distributed as binomial. The parameter *n* is the number of binomial trials per case, and *p* is the probability of "successes." For random assignment, $n = 1$, and *p* depends on the training/test breakdown you want. Setting *p* to .5 will generate a new variable with a 50%–50% breakdown of ones and zeros; $p = .7$ produces a 70%-30% breakdown, and so on.

Independent testing is another story. As far as we know, this would involve building a model (e.g. a logistic regression) on one part of the data (where `x == 1`), saving parameter estimates in a vector, and then predicting the outcome in the rest of the data (where `x == 0`) using a regression equation generated through multiplying a matrix of variables by this vector of coefficients. This process is somewhat arduous, and gets the user into the field of matrix programming in Stata, which is fairly advanced. In short, holdback validation is difficult to do in Stata, principally because its designers haven't built the system with data mining concerns in mind.

K-fold CV can be done straightforwardly with a user-generated program called crossfold (Daniels 2012). Crossfold uses the following Stata syntax:

```
crossfold regress yvar xvars, k(k)
```

"`Regress`" in this syntax could be replaced with logit or other estimators. We are not sure, and documentation does not tell us, precisely how many estimators are supported by crossfold. In any event, it provides fit statistics from *k* models (with *k* chosen by the researcher), and permits the choice of fit statistics—RMSE, mean absolute error, or pseudo-R^2. Typical output is shown in table 5.1. This is not the most informative CV technique, but it does provide *k* independent tests of a model.

R

We are not aware of a specific R routine which performs holdback CV and which could be combined with any analytical routine. There are some R routines which *incorporate* CV via holdback (presuming one has already created separate test and training sets). For these routines, holdback CV is quite easy; for others, it is as difficult as it is in Stata.

However, numerous *k*-fold CV routines exist in R. One *k*-fold CV routine which is particularly intuitive is cv.glm, which is part of the package called *boot* (Canty and Ripley 2010). It is used to cross-validate previously fit generalized linear models.

One can call this with the following syntax:

```
cv1<-cv.glm(data, glmfit, k)
```

Where `data` is some dataset, `glmfit` is the results of a previously fit generalized linear model fit using `data`, and `k` is the number of folds desired. After running this model, entering

```
cv1$delta
```

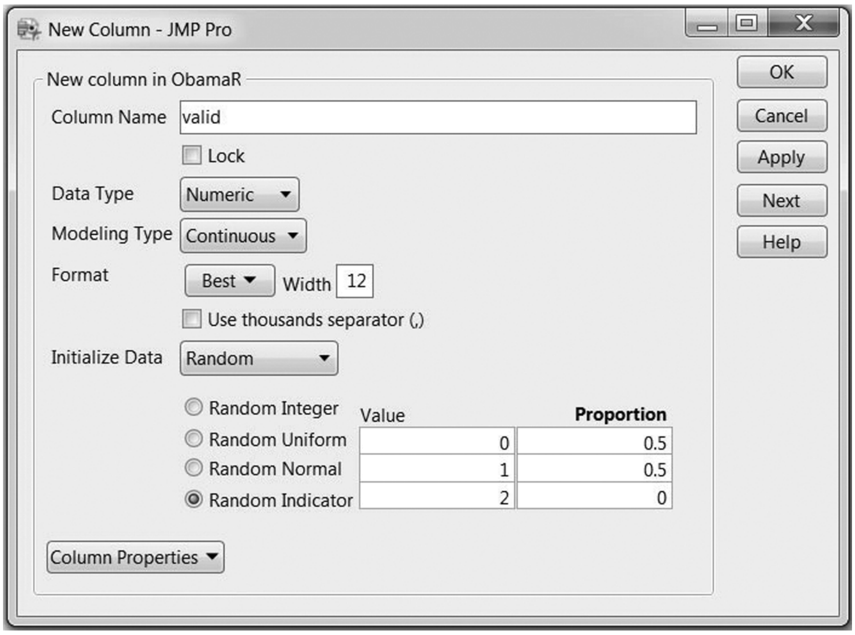
returns a vector of two numbers: a cross-validation fit, and an adjusted cross-validation fit (if a value of *k* was entered rather than using the default leave-one-out CV).[2] The latter number is given because some statistical work suggests that *not* using leave-one-out CV can generate biased estimates of validation fit (Davison and Hinkley 1997), so the program performs some operations to compensate for this bias.

JMP PRO

CV is very, very easy in JMP Pro. It is done in one of two ways. For some modeling routines, JMP Pro requires you to supply a validation variable—a nominal variable containing different values indicating the training and test portions. What we want is a variable in which the values are randomly assigned in the proportions we desire. This can be done quite easily. In the main menu of JMP Pro, click Cols New Column. The window which opens is shown in figure 5.1.

We change the column name from the default to "valid" (as in validation). We change Modeling Type from the default Continuous to Nominal, and then under Initialize Data, we select Random. This permits us to choose different distributions to draw from: the normal, the uniform, and the binomial (which is what Random Indicator is drawing from). We select Random Indicator and change the proportions for values 0 and 1 to 0.5 and 0.5 (the value for 2 is set to 0). Simply clicking OK proceeds to generate this validation variable. The variable can simply be entered into Validation fields in later model-building windows, as in the Stepwise Regression launch platform shown in figure 5.2.

Other model platforms have built-in CV settings in JMP. The Partition platform (for partition trees), for example, allows the researcher to indicate a "validation portion" in the main model platform window. They also allow users to choose *k*-fold CV in the model launch window. Neural nets similarly allow users to specify a holdback portion. We will go into this in greater detail in the partition tree and neural network sections below.


New Column - JMP Pro

New column in ObamaR

Column Name: valid

Lock: unchecked

Data Type: Numeric

Modeling Type: Continuous

Format: Best, Width 12

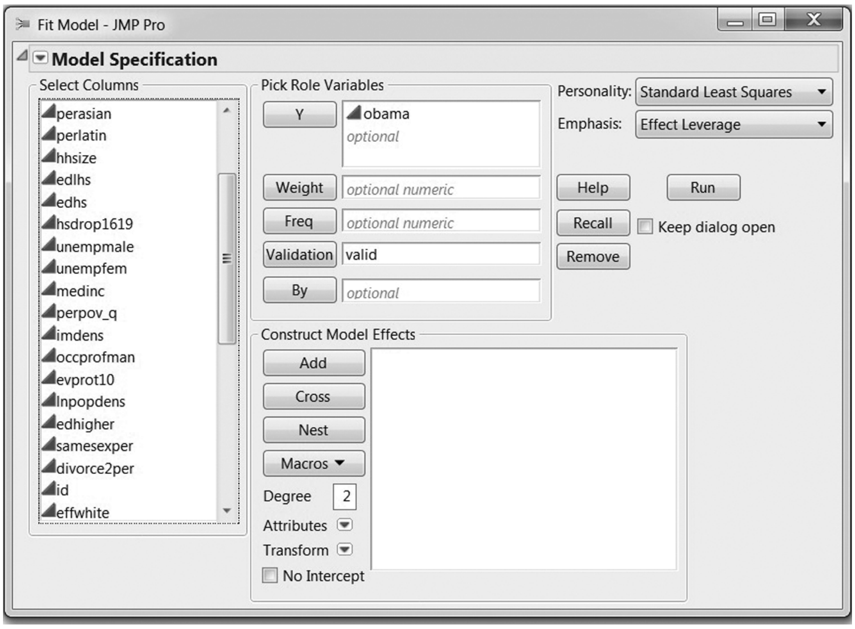
Use thousands separator (,): unchecked

Initialize Data: Random

Random Integer / Random Uniform / Random Normal / [✓ Random Indicator]

Value	Proportion
0	0.5
1	0.5
2	0

FIGURE 5.1
Creating a validation column in JMP Pro.


Fit Model - JMP Pro

Model Specification

Select Columns: perasian, perlatin, hhsize, edlhs, edhs, hsdrop1619, unempmale, unempfem, medinc, perpov_q, imdens, occprofman, evprot10, lnpopdens, edhigher, samesexper, divorce2per, id, effwhite

Pick Role Variables

Y: obama

Weight: optional numeric

Freq: optional numeric

Validation: valid

By: optional

Personality: Standard Least Squares

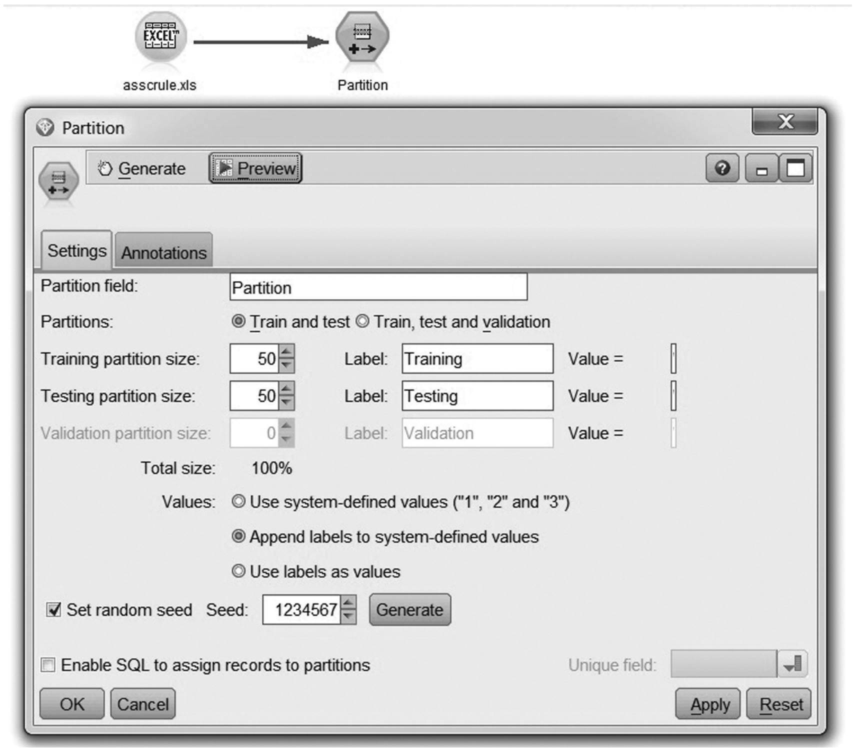
Emphasis: Effect Leverage

Keep dialog open: unchecked

Construct Model Effects: Degree 2

No Intercept: unchecked

FIGURE 5.2
Adding cross-validation to stepwise regression in JMP Pro.



Partition field: Partition

Partitions: [✓ Train and test] Train, test and validation

Training partition size: 50 Label: Training Value =

Testing partition size: 50 Label: Testing Value =

Validation partition size: 0 Label: Validation Value =

Total size: 100%

Values: Use system-defined values ("1", "2" and "3")

[✓ Append labels to system-defined values]

Use labels as values

[✓ Set random seed] Seed: 1234567 Generate

Enable SQL to assign records to partitions Unique field:

FIGURE 5.3
Cross-validation in SPSS Modeler.

SPSS MODELER

SPSS Modeler, the specialized data analytics package of SPSS, makes CV by holdback particularly easy. The program, which we will describe in more detail later, involves generating "streams" of statistical operations through point-and-click; each stream consists of "nodes" which can perform operations. Each node has an associated window in which numerous parameters can be adjusted. The nodes are selected from "palettes" which contain similar-type nodes.

In the Fields palette, choose the Partition node and double-click. This opens the screen shown in figure 5.3. Within this node, as can be seen, one can set the breakdown of training and test (and validation) samples. This generates a variable called Partition (or some other name if the researcher changes it) which can be selected as a validation variable in modeling nodes.

K-fold CV can be also performed for some other applications in Modeler (*K*-nearest neighbors, 5.0 partition tree, neural net), but within the nodes for these specific modeling operations, not as a separate node.

SPSS STATISTICS

IBM has been enhancing SPSS Statistics—the regular statistics program that is used in hundreds of undergraduate classrooms—with several data mining applications. Some of those have internal options for CV. However, it is easy enough to divide any SPSS dataset into two random parts—for training and testing—using ordinary SPSS syntax. In the example below we divide our data randomly such that 80% of cases are identified as training and the remaining 20% as test; users can choose their own proportions. The syntax reads:

```
USE ALL.
COMPUTE filter_$ = (uniform(1)< = .80).
VARIABLE LABELS filter_$ ‘Approximately 80% of the cases
  (SAMPLE)’.
FORMATS filter_$ (f1.0).
FILTER BY filter_$.
EXECUTE.
Filter OFF.
recode filter_$ (0 = 0)(1 = 1) into datagroup.
var label datagroup ‘training or test’.
value labels datagroup 0 ‘test dataset’ 1 ‘training data-
  set’.
execute.
```

The variable “datagroup” takes values of 1 for observations randomly assigned to the 80% training subset and 0 for those to be used for the test dataset.

6

VARIABLE SELECTION TOOLS

When analyzing big data, we are faced with an overwhelming welter of information. We have too many cases, or too much information about each case, for effective use of standard statistical methodologies. We have already seen how having too many cases can cause programs to crash or run unduly slowly, and how this can sometimes be addressed simply by sampling our data. A more complicated situation emerges when we have too much information gathered about each case—in other words, when we have more variables than we know what to do with.

Data miners use the letter *N* to refer to the number of observations or cases, and the letter *P* to refer to the number of variables, predictors or features. In the situation where *P* is very large, what we seek is a technique that reduces the amount of information we need to process by selecting those variables which are most important and discarding the others.

One solution is to turn to a class of techniques called *subset selection* or *regularization* methods. This family of methods were developed precisely in order to automate the process of variable selection (and for this same reason it is often criticized by conventional data analysts for being atheoretical and data-driven). The methods in question operate by discovering, from among a long list of predictors, those which perform best in terms of explaining variance on the dependent variable.

STEPWISE REGRESSION

Stepwise regression was the first "automated" routine that was developed for choosing variables, and it is certainly the one most frequently and vehemently denounced by detractors of such algorithms. It was developed for situations in which a researcher is blessed with an abundance of independent variables in a dataset (i.e., large *P*) but has little or no theoretical framework for choosing which of them to include in a model. This is probably still the situation in which it is most often employed, but there are other situations in which it can be profitably used and for which it is probably more justifiable.

In our experience, stepwise regression may be used not only to weed through large numbers of potential predictors representing main effects but also to sift through higher-order interaction terms between predictors. Let's imagine we have 12 predictor variables which we want to include in a model. But we want to be realistic about the fact that the world does not consist only of main effects but also of interactions between predictors, and we want to allow for the possibility that some of the interactions might explain substantial variance in the outcome variable. If we wish to include only two-way interactions, the number of features or predictors in the model rises from 12 to 88. That number increases to 100 if we also include quadratic terms—interactions of a variable with itself—to allow for curvilinear relations between a predictor *X* and an outcome *Y*. If we also decide to allow for three-way interactions, (say of age by gender by income) the number of predictors in the model reaches 320 terms. However, not all of those 320 variables are likely to be statistically significant or important predictors. So we begin to see the appeal of a feature selection algorithm which can tell us which among these 320 variables are important.

Stepwise regression routines work in one of three ways. The first, *forward selection,* begins with a model that contains only an intercept term. It then examines each independent variable individually and chooses the one that is "best" (we will get to how "best" is determined in a minute). After entering this variable into the predictive model, the program repeats this process, considering the remaining candidate predictors over and over—adding one superior predictor at a time—until it chooses the "best" model (once again, we'll define that a little later).

The second method, *backward elimination,* begins by including all available variables in an initial regression, and then tests each one to see which can be most profitably dropped from the model. It ends with a more pared-down model in which only the variables that "count" remain. Finally, there is a method known as *forward-backward stepwise regression,* which as the name suggests is a combination of forward selection and backward elimination. Like forward selection it starts with a null model and enters variables iteratively when they meet a criterion, but it also eliminates them if and when they subsequently fall below some threshold of relevance.

Variables are selected for inclusion or exclusion in one of two ways. The first involves the use of *p*-values for the individual predictor variables. For example, the researcher might instruct the program to include variables only if they have p-values of .05 or lower and to eliminate them if their p-value rises above .10. This variable inclusion criterion, which is oriented entirely toward the individual predictors, is what is used exclusively in the stepwise algorithms of some commercial packages—Stata's, for instance.

An alternative approach is to employ some measure of overall or global model fit, usually one that penalizes a model for adding more predictors. Fit measures include adjusted R^2, Akaike's information criterion (AIC), the Bayesian information criterion (BIC), and Mallows's C_p. Variables are included or excluded on the basis of the improvement each makes to the regression model as a whole as evaluated by the change in the fit statistic designated by the researcher.

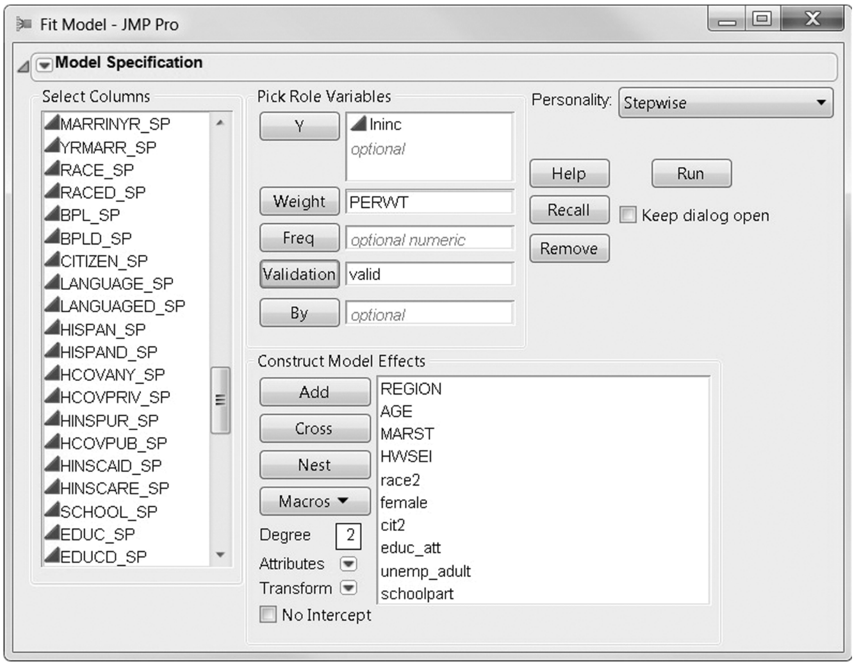
The "final model" is chosen in a manner similar to how the individual variables are chosen. If p-values are utilized in variable selection, then the algorithm will stop building a stepwise regression model once all of the variables that meet the researcher-specified criteria are in the model (for instance, all variables in the model have p-values of .05 or lower, and no other variable could be entered into the model which would have such a p-value). If, on the other hand, a global fit statistic is utilized, the algorithm will choose the model which optimizes that fit statistic—that is, the model with the highest adjusted R^2 or the lowest BIC, for example.

Both methods are quite susceptible to type I error, because by chance alone, we are quite likely to find, in a large set of predictors, one that is significantly related to the outcome. While this may be rather obvious in the case of p-value-based methods of selection, it is also true of selection methods that use global measures of model fit. Given enough predictors, one is bound, by chance, to be able to raise predictive capacity sufficiently to surmount the threshold for inclusion. The best ways to avoid type I error appear to be those which use a measure of model fit as a "stopping rule" and which select variables on the basis of p-values—but p-values which take the multiple-comparisons issue into account. We could, for instance, apply the Bonferroni rule, setting the p-value at α/p, where p is the total number of candidate predictors and α is 0.05. Such Bonferroni p-values are very stringent, though. Another approach, suggested by Foster and Stine (2004), is to test variables in ascending order according to their t statistics, beginning conservative threshold and gradually raising that threshold while working towards the less predictive variables.

Like any other data-driven approach, stepwise regression is highly likely to overfit the model in question (though using a tougher threshold for inclusion will remedy this to some extent). Validating the model on a separate test set of data is therefore important. When possible, the cross-validation fit should be reported.

AN EXAMPLE IN JMP

We will demonstrate the utility of stepwise regression using JMP's Stepwise algorithm, which we like because of the numerous options for stopping rules that it provides and because of the ease with which interactions and polynomial terms can be added to the


Fit Model - JMP Pro

Model Specification

Select Columns: MARRINYR_SP, YRMARR_SP, RACE_SP, RACED_SP, BPL_SP, BPLD_SP, CITIZEN_SP, LANGUAGE_SP, LANGUAGED_SP, HISPAN_SP, HISPAND_SP, HCOVANY_SP, HCOVPRIV_SP, HINSPUR_SP, HCOVPUB_SP, HINSCAID_SP, HINSCARE_SP, SCHOOL_SP, EDUC_SP, EDUCD_SP

Pick Role Variables

Y: lninc

Weight: PERWT

Freq: optional numeric

Validation: valid

By: optional

Personality: Stepwise

Keep dialog open

Construct Model Effects: REGION, AGE, MARST, HWSEI, race2, female, cit2, educ_att, unemp_adult, schoolpart

Degree: 2

No Intercept

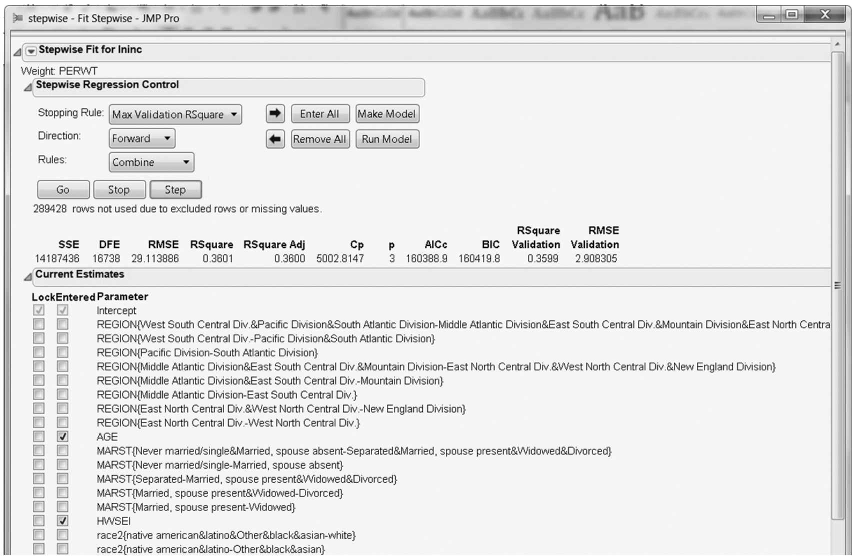
FIGURE 6.1
Stepwise regression in JMP Pro.

model. (However, stepwise regression is available in many other statistical packages, including SPSS and SAS.)

For this demonstration, we will use data from the U.S. Census Bureau's 2010 American Community Survey, from which we have randomly drawn a dataset of 15,000 cases. We will be predicting the (logged) personal income of adults in the United States using a fairly small number of main effect variables—region, age, marital status, occupational prestige, race, gender, citizenship, educational attainment, employment status, and educational enrollment.

To run a stepwise regression using JMP, we go to Analyze and choose Fit Model (figure 6.1). In the top-right corner of the Fit Model dialog box, we click the Personality menu and choose Stepwise. We can add a probability weight (a variable called PERWT provided by the survey to correct for nonresponse). We also tell the program the name of a Validation variable, which we called "valid" and had previously created in JMP, and which randomly divides the dataset into a training dataset and a test set (in a 2:1 ratio). Next, we click Run. This opens the Stepwise launch platform (figure 6.2), which lists all the terms or variables we are including in our model.

We should explain that JMP does something clever with non-dichotomous categorical variables in stepwise regression. Rather than representing the categorical variable as a



```
Stepwise Fit for lninc
Weight: PERWT
Stepwise Regression Control
  Stopping Rule: Max Validation RSquare
  Direction:     Forward
  Rules:         Combine
289428  rows not used due to excluded rows or missing values.

                                                                                  RSquare       RMSE
     SSE      DFE       RMSE  RSquare  RSquare Adj         Cp  p      AICc       BIC  Validation  Validation
14187436    16738  29.113886   0.3601       0.3600  5002.8147  3  160388.9  160419.8      0.3599    2.908305

Current Estimates
LockEntered Parameter
 ✓    ✓     Intercept
 ☐    ☐     REGION{West South Central Div.&Pacific Division&South Atlantic Division-Middle Atlantic Division&East South Central Div.&Mountain Division&East North Centra
 ☐    ☐     REGION{West South Central Div.-Pacific Division&South Atlantic Division}
 ☐    ☐     REGION{Pacific Division-South Atlantic Division}
 ☐    ☐     REGION{Middle Atlantic Division&East South Central Div.&Mountain Division-East North Central Div.&West North Central Div.&New England Division}
 ☐    ☐     REGION{Middle Atlantic Division&East South Central Div.-Mountain Division}
 ☐    ☐     REGION{Middle Atlantic Division-East South Central Div.}
 ☐    ☐     REGION{East North Central Div.&West North Central Div.-New England Division}
 ☐    ☐     REGION{East North Central Div.-West North Central Div.}
 ☐    ✓     AGE
 ☐    ☐     MARST{Never married/single&Married, spouse absent-Separated&Married, spouse present&Widowed&Divorced}
 ☐    ☐     MARST{Never married/single-Married, spouse absent}
 ☐    ☐     MARST{Separated-Married, spouse present&Widowed&Divorced}
 ☐    ☐     MARST{Married, spouse present&Widowed-Divorced}
 ☐    ☐     MARST{Married, spouse present-Widowed}
 ☐    ✓     HWSEI
 ☐    ☐     race2{native american&latino&Other&black&asian-white}
 ☐    ☐     race2{native american&latino-Other&black&asian}
```

FIGURE 6.2
Output from stepwise regression in JMP Pro.

set of zero-or-one dummy variables, with one baseline category omitted, JMP codes categories hierarchically. It first splits the categories into the two groups which have the most disparate means for the response variable, and adds a dummy variable for this contrast. Within these two groups, it then splits them again into two more groups in the same manner, and so on. For example, consider what JMP does with the "educational attainment" variable. It splits first into two groups: *less than high school* versus all other categories. Next, it splits the latter group into *high school + some college + no degree* versus *associate's + bachelor's + higher than bachelor's*. The former group it can only split once, but the latter group again splits into *associate's + bachelor's* versus *higher than bachelor's*. The point is that these groups are clustered hierarchically in terms of the outcome, such that groups with more similar means on the outcome variable will remain together in one group. This hierarchical differentiation has the benefit of permitting the program to choose a more parsimonious regression model than one that includes all the separate values of a categorical variable as dummies, if that parsimony is beneficial to model fit.

As shown in the screenshot (figure 6.2), we have specified a validation variable, and so we will use maximum validation R^2 as the stopping rule. Other options are *k*-fold cross-validation R^2 (available only if a validation variable is not specified), *p*-value thresholds (to enter and to leave the model), minimum AIC, and minimum BIC. Next, we specify what direction the stepwise algorithm should proceed in. We choose forward stepwise regression. Other options are backwards and mixed, the latter available only when *p*-value stopping rules are utilized.

The Rules menu relates to our hierarchically ordered categorical variables. In the default setting, Combine, as well as with the "Restricted" option, the selection of a more fine-grained differentiation will trigger the automatic inclusion of higher-level groupings. If you don't want this, change the setting to No Rules (which we strongly advise against). Incidentally, models with interaction terms operate in a similar fashion in JMP's stepwise algorithm. That is, the selection of an interaction will automatically lead to the inclusion of all constituent terms, unless No Rules is selected.

JMP permits you to run the program one step at a time, to watch the progress of the model as it evolves. The first step in the main-effects-only model includes occupational prestige, which achieves an R^2 of .21. The next step includes age, which raises the R^2 to .36, followed by an indicator for unemployed working-aged adults ($R^2 = .44$). When we permit the stepwise regression to run to the end and to choose the best model for predicting log income, the algorithm proceeds to select 24 parameters, and achieves a validation R^2 of .495, as shown in table 6.1. The progress of model fit can be seen by choosing, in the top corner of the output window, the "red triangle" menu, Criterion History, and then RSquare History. (Many of JMP's windows contain menus, styled as red triangles pointing down. Though this book is printed in black and white, we will still refer to these menus as red triangles.) As shown in figure 6.3, most of the improvement in model fit was achieved in the first 10 steps, and only very modest improvements were made after 20 steps or predictors. Nonetheless, the fit did continue to improve incrementally afterwards.

We can certainly do better than this in terms of fit if we include interaction terms. This is done by choosing Relaunch Model, which returns us to the Fit Model box. We now set Degree to 2 (for two-way interaction terms). In the Select Columns box, we highlight all of our variables, and then from the Macros menu we choose Factorial to Degree. This automatically enters all possible two-way interactions as candidate variables. For good measure, we also include quadratic terms for age and occupational prestige. This time, the program takes 54 steps to build the optimal model in terms of validation fit. The selected model now has an R^2 of .6123 in the training set and .6064 in the test set, and has entered a total of 68 parameters in the model, including the interaction terms.

Many of the terms selected were interactions, which trigger the inclusion of constituent terms. Thus, all main-effect variables were used to some degree in the model, but not all separate categories of nominal or categorical variables were utilized. For example, only three region contrasts were included, and the educational attainment variable was not thoroughly disaggregated.

With a model this complex, in which many interactions were utilized, along with groupings of categorical variables, interpretation of the parameters is difficult. Consider the age parameter. The model has selected the age main effect, the quadratic term, and eight interaction terms involving age. Here there is a strong trade-off between predictive accuracy and interpretability. With this level of complexity, it is difficult, though certainly

AU: Please provide a column head for the leftmost column, if possible.

TABLE 6.1 Main Effects Regression Model

	β	p
Region: West South Central, South Atlantic and Pacific vs. else	–0.030	.001
Region: West South Central vs. South Atlantic and Pacific	–0.097	.001
Region: South Atlantic vs. Pacific	0.075	.028
Region: Mid Atlantic, East South Central, and Mountain vs. East North Central, West North Central, and New England	–0.023	.049
Region: East North Central and West North Central vs. New England	0.038	.104
Region: East North Central vs. West North Central	0.085	.071
Age	0.040	<.001
Marital status: never married and married, spouse absent vs. separated, married, spouse present, widowed and divorced	–0.229	<.001
Marital Status: separated vs. married, spouse present, widowed and divorced	0.062	<.001
Marital Status: married, spouse present and widowed vs. divorced	–0.133	<.001
Marital Status: married, spouse present vs. widowed	–0.189	<.001
Occupational prestige	0.061	<.001
Race: else vs. white	0.046	<.001
Race: Native American and Latino vs. black, Asian, and other	0.193	<.001
Race: other and black vs. Asian	0.261	<.001
Female	–0.281	<.001
Noncitizen	–0.110	.007
Education: less than high school vs. rest	–0.639	<.001
Education: high school and some college vs. AA, BA, and higher	0.141	<.001
Education: high school vs. some college	–0.146	<.001
Education: AA and BA vs. higher	0.193	<.001
Unemployed	–1.435	<.001
In school	–1.039	<.001
Constant	3.204	<.001
R^2	0.508	
Validation R^2	0.495	

not impossible, to interpret just what the "effect of a one-unit change in age on income" would be. However, including these interaction terms did increase out-of-sample predictive accuracy.

SUMMARY

Stepwise regression can be used to select, from a large set of independent variables, those that are most predictive. It is typically used in contexts where a researcher has many

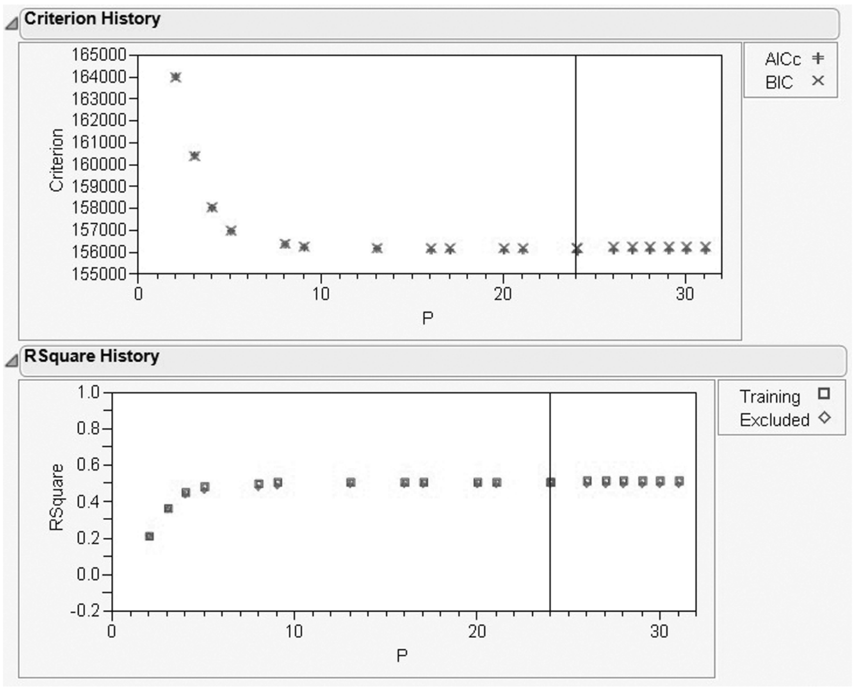


FIGURE 6.3
The progress of stepwise model fit in JMP Pro.

features in the dataset. The technique can also be used to identify those interaction terms between predictors that enhance the predictive power of a model. There are usually many possible interaction terms, and stepwise regression can be used to discover the useful ones.

It is best to use stepwise regression with some form of cross-validation, because this technique will overfit data. The R^2 or fit statistic for the training data will therefore be artificially high. However, if one attends to the predictive power of a model for the test sample, then overfitting ceases to be a problem. The R^2 for the test dataset is a valid measure of the predictive strength of the regression model.

THE LASSO

As mentioned, stepwise regression, like many data mining approaches, is disparaged by some data analysts who view the method as atheoretical "data fishing" or "data dredging." Such researchers feel that statistical modeling should always be driven by theory about causal processes and the variables that represent them. But stepwise regression has also been criticized from within the friendly terrain of data mining research itself. These critics note that the all-or-nothing nature of its variable selection process—variables are either included or left out—makes stepwise regression unstable and therefore somewhat unreliable. Small changes in the data, such as different random samplings from a larger set of cases, can lead to different subsets of variables being chosen by the stepwise algorithm. Therefore, a method that, rather than keeping or discarding variables wholesale, instead enacts selection more gradually and continuously seems preferable.

The LASSO (which stands for the least absolute shrinkage and selection operator) is just such a method. It imposes a penalty on the model's regression coefficients in such a way that those of the less predictive variables are shrunk toward zero. At the point where the regression coefficients reach zero, they effectively leave the model. This makes the LASSO similar in form to the backwards elimination form of stepwise regression. But unlike backwards elimination, model simplification in the LASSO does not occur through the establishment of an arbitrary threshold. The gradual nature of the LASSO's shrinkage process means that the inclusion or exclusion of one variable does not immediately and profoundly affect the coefficients of those that remain. The LASSO is thus more stable and produces less bias than stepwise regression, but still results in similar model simplification.

Mathematically, the LASSO's penalty relates to the sum of the absolute values of the regression coefficients (this is the *Manhattan* or *city-block distance* of the vector of coefficients, also called their *L_1 norm*), usually after all predictors have been standardized as z-scores. In its initial formulation (Tibshirani 1996), the sum of these absolute values was constrained to be less than a tuning parameter, t. If t is set to be greater than or equal to the observed sum of the absolute values of coefficients from the baseline OLS model, no shrinkage would occur and the LASSO fit would equal the least-squares fit. Shrinking this tuning parameter below that sum, then, has the effect of constraining these coefficients. Other variants, like the method we use, use a transformation of this parameter which increases constraints more at higher values.

To demonstrate, we begin by presenting a full OLS regression model predicting vote percentage for Obama in 2012 in U.S. counties. This "ordinary" regression is going to be used as a benchmark against which to compare the LASSO. We have picked a fairly large set of independent variables here—22 total—which describe several demographic dimensions of these counties (population density, racial mix, age structure, economic characteristics, etc.). Table 6.2 shows the results of this regression. To be sure, this model is of some interest in its own right, and it explains a good percentage of the variance in voting: 58%. But clearly we have some predictors which are related, and we may be interested in a more stripped-down, parsimonious model. This is an excellent opportunity to make use of the LASSO.

There is a user-generated Stata program which implements the LASSO ("lars," specifying the "lasso" function), but it appears to be in the early stages of development. The capacity for the LASSO also exists in JMP Pro 12 and in SPSS Statistics, as long as one has purchased the SPSS Categories package. R has at least two routines which perform the LASSO. They are called "penalized" and "lars," and both are freely available through

TABLE 6.2 Results from an OLS Regression Predicting Obama's Vote Share in County-Level Data

Variable	Coeff. (SE)	Standardized (Beta) Coeff.
Population density (log)	2.398 (0.157)***	0.278
% under 18 years	–0.775 (0.101)***	–0.177
% 18–34 years	–0.534 (0.0710)***	–0.177
% 65 and older	–0.636 (0.100)***	–0.179
% non-Hispanic white	–0.476 (0.0311)***	–0.625
% Asian	0.165 (0.116)	0.0260
% non-Hispanic black	0.0147 (0.0309)	0.0145
% Latino/a	–0.0748 (0.0360)**	–0.0651
% college graduate	0.563 (0.0613)***	0.329
% high school graduate only	0.314 (0.0477)***	0.147
% less than high school education	–0.214 (0.0514)***	–0.106
Male unemployment rate	0.920 (0.0729)***	0.232
Female unemployment rate	–0.0808 (0.0795)	–0.0184
Poverty rate	0.249 (0.0642)***	0.101
% foreign-born	–0.174 (0.0643)***	–0.0637
% divorced	–0.104 (0.0424)**	–0.0438
% same-sex-couple households	1.718 (0.582)***	0.0374
% evangelical Protestant	–0.277 (0.0133)***	–0.304
Average household size	–8.242 (1.341)***	–0.135
% professional/managerial	–0.369 (0.0487)***	–0.163
Median income	–7.34e-05 (4.16e-05)*	–0.0567
High school dropout rate	–0.0173 (0.0337)	–0.00655
Constant	130.0 (7.903)***	
Observations	3,114	
R^2	0.586	

I J OE: Standard errors in parentheses.
***p < .001; **p < .01; *p < .05.

cran.r-project.org. We will assume a baseline familiarity with R and focus here on the "penalized" package (Goeman, 2010; Goeman, Meijier & Chaturvedi 2012).

We begin with a simple call of the R function, involving mostly default settings of the program:

```
lasso1<-penalized(obama, ~lnpopdens+agelt18+age1834+age65o
  ver+imdens+perwhite+perasian+perblack+perlatin+edhigher+
  edhs+edlhs+unempmale+unempfem+perpov_q+divorce2per+sames
  exper+evprot10+hhsize+occprofman+medinc+hsdrop1619,
  lambda1 = 500, standardize = TRUE)
```

This runs the model. The `lambda1` option sets the penalty relating to the sum of absolute values of coefficients; larger values will result in more shrinkage toward zero of regression coefficients. It is also possible to utilize a separate option called `lambda2` which relates to the square root of the sum of squares of regression coefficients (their Euclidean distance or L_2 *norm*). Including lambda 2 instead of lambda 1 thus leads penalized to perform *ridge regression*. It is possible in penalized to set both lambda1 and lambda2 in order to penalize the model in a more complex fashion. One may also exclude certain covariates from penalization, and may penalize the various coefficients differently. But we will focus here on a straightforward case of the LASSO. We have also standardized our variables as *z*-scores in advance (with `standardize = TRUE`).

To see the regression coefficients, we enter:

```
coefficients(lasso1, 'all')
```

Table 6.3 shows our results. We initially set a low penalty parameter here (a value of 2 for `lambda1`): see the column labeled 2. Therefore, the variables all have non-zero coefficients and are nearly identical to the OLS estimates. In fact we have to make the penalty much higher to see substantial model alteration. Nothing at all drops out until the penalty reaches 500. In the 500 column, some of the coefficients for predictors have dropped to zero. Even after doubling the penalty again (to 1,000), we retain 15 covariates. This is undoubtedly because many of our covariates actually contribute to explaining variance in the outcome, and because of our relatively large ($N = 3{,}114$) sample size.

Once the variables start being shrunk to zero, however, some interesting things happen. While most coefficients shrink monotonically as the penalty increases, the coefficient on percentage black actually rises until `lambda1 = 1,000`, after which it drops off only slightly. The small coefficient on black population share in the initial model had been surprising; this suggests that in a multivariate model the effects of this variable are largely masked by related covariates, but that it is an important predictor in and of itself. Another variable, divorced people as a percentage of ever-married adults, drops to zero at `lambda1 = 500`, re-emerges at 1000, and then again shrinks to zero. In the final column of the printout, with a penalty of 5,000, we have a much smaller set of covariates to examine, each of which explains a fair amount of the variance in the data.

Penalized can also produce a graph showing how the regression coefficients shrink as the penalty increases. To see this graph, we first tell the program to calculate the coefficients as it increases the penalty in regular intervals (`steps = 100`). While it is entirely possible to draw the graph using a large number of variables as in the above model, the resulting graph will be a touch overcrowded. For clarity of presentation, we estimate a simpler model:

```
lasso1<-penalized(obama, ~lnpopdens+imdens+perblack+perwhi
  te+edhigher+evprot10, lambda1 = 2, steps = 100, trace =
  FALSE, standardize = TRUE)
```

TABLE 6.3 Regression Coefficients from LASSO Predicting 2012 Obama Vote Share, with Varying Penalties

	Value of lambda1 penalty				
	2	*100*	*500*	*1000*	*5000*
Population/sq. mile (log)	2.396	2.313	2.139	2.063	1.631
% age < 18	−0.7730	−0.689	−0.474	−0.375	−0.131
% non-Hispanic white	−0.4756	−0.452	−0.372	−0.300	−0.172
% black	0.01499	0.027	0.079	0.125	0.116
% BA or higher degree	0.5619	0.527	0.402	0.246	0.166
Male unemployment rate	0.9195	0.916	0.973	1.025	0.699
% evangelical Protestant	−0.2767	−0.274	−0.263	−0.249	−0.163
% Asian	0.1646	0.150	0.101	0.0301	0.000
% HS diploma	0.3139	0.301	0.233	0.126	0.000
% less than HS	−0.2140	−0.200	−0.171	−0.158	0.000
Poverty rate	0.2489	0.238	0.153	0.070	0.000
% same-sex households	1.716	1.644	1.504	1.355	0.000
Avg. household size	−8.242	−8.263	−7.425	−5.926	0.000
% prof./managerial	−0.3682	−0.333	−0.201	−0.053	0.000
Divorce rate	−0.1036	−0.062	0.000	0.012	0.000
% age 18–34	−0.5319	−0.439	−0.161	0.000	0.000
% age 65+	−0.6331	−0.509	−0.177	0.000	0.000
% foreign-born	−0.1735	−0.159	−0.087	0.000	0.000
% Latino	−0.07460	−0.063	−0.026	0.000	0.000
Female unemployment rate	−0.07988	−0.036	0.000	0.000	0.000
Median income ($1000s)	−0.073	−0.030	0.000	0.000	0.000
HS noncompletion rate	−0.01715	−0.010	0.000	0.000	0.000

—and we tell it to create a plot of this relation. It helps to standardize the coefficients here so that their different scales do not lead to some being overwhelmed by others:

```
plotpath(lasso1, log = ”x”, standardize = TRUE)
```

The resulting graph (figure 6.4), along with standardization, lets us visualize which variables remain important to the model. We can see that the variables that remain in the model longest are those which, at the beginning, were most highly correlated (positively or negatively) with Obama’s share of the vote: the percentage of county population that is non-Hispanic white, population density, male unemployment rate, the percentage of the population that is evangelical Protestant, and the percentage of the adult population with a bachelor’s degree or higher. A regression model with only these five variables has an adjusted R^2 of .49, compared to .58 in the full model. Clearly these five variables can tell us quite a bit about aggregate electoral patterns.

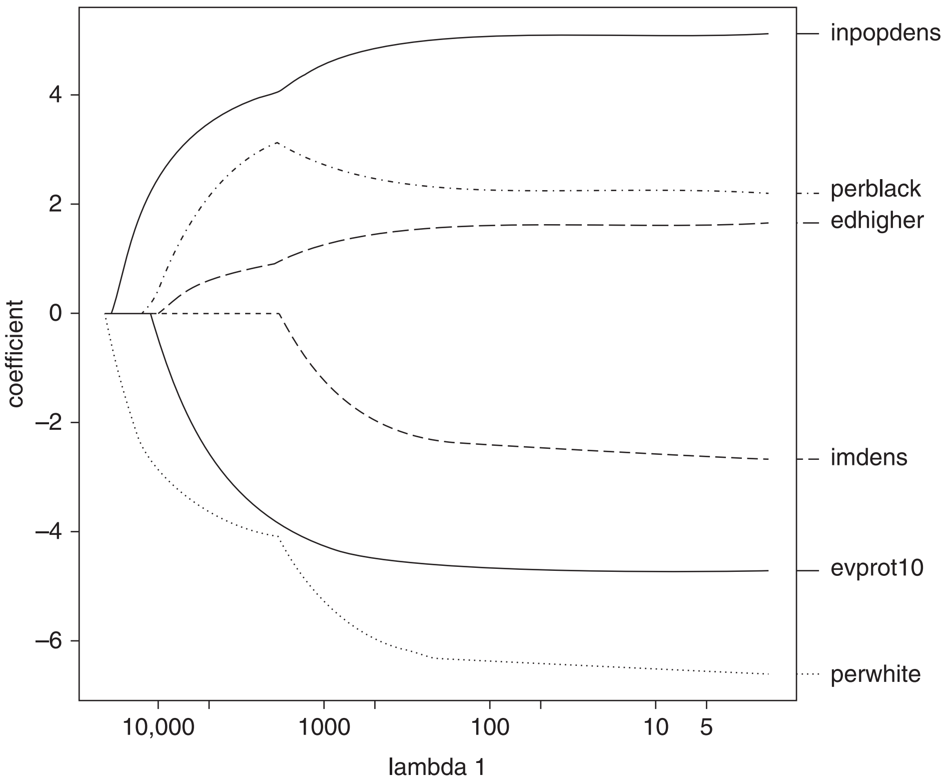


FIGURE 6.4
Shrinkage of coefficients at different settings of the penalty parameter in LASSO (from R).

However, dropping variables from the model, though making the model substantially more interpretable, does reduce the total predictive value. Typically, we want to balance parsimoniousness with predictive accuracy.

The penalized package also allows us to use *k*-fold cross-validation to determine how well a model fits. We can perform cross-validation of a model with the following code:

```
cross<-cvl(obama, ~lnpopdens+agelt18+age1834+age65over+imd
  ens+perwhite+perasian+perblack+perlatin+edhigher+edhs+ed
  lhs+unempmale+unempfem+perpov_q+divorce2per+samesexper+e
  vprot10+hhsize+occprofman+medinc+hsdrop1619, lambda1 =
  500, fold = 10, standardize = TRUE)
```

Here we create an object called "`cross`" in which is stored cross-validation results from a model predicting Obama's vote share using 22 independent variables. After the model, we must set the penalty parameter (`lambda1 = 500`) and the number of folds to be used in cross-validation (`fold = 10`). After running the model, we call elements

TABLE 6.4 Fit Statistics at Different Values of the Penalty Parameter in the LASSO

	Value of lambda1 penalty				
	2	*100*	*500*	*1,000*	*5,000*
Cross-validation log-likelihood	–11,483.17	–11,496.82	–11,516.6	–11,568.63	–11,888.41
Full data log-likelihood	–11,443.92	–11,445.92	–11,475.19	–11,533.25	–11,827.07
Nonzero coefficients	23	23	19	16	8

of the object. The first element, `$cvl`, returns the log-likelihood of the model on the cross-validation data. With the element `$fullfit`, we can call the fit on the full data.

```
cross$cvl
cross$fullfit
```

We can repeat this a number of times at different settings of lambda1, and examine the relative fit of the model with different penalty parameters. In table 6.4 we show cross-validation log-likelihoods for the above model, with the values of lambda1 we used above to demonstrate model shrinkage. We see that the lowest log-likelihood statistics listed are at `lambda1 = 2`. This would seem to suggest that less, not more, shrinkage yields a better fit in this model. But these few results alone do not allow us to conclude that 2 is the best value for lambda for maximizing out-of-sample accuracy. To do this, we would have to proceed like a maximum-likelihood algorithm, trying different values and moving closer and closer to the minimum cross-validation log-likelihood.

This sounds like a lot of work which ought to be automated. And luckily, it has been. Penalized will permit us to find the best value for lambda1 with the '`optL1`' function. This function allows us to set minimum and maximum values of lambda1, and it will find the value that minimizes the cross-validation log-likelihood. We set the minimum to zero, and allow lambda1 to range up to 1,000:

```
bestfit<-optL1(obama, ~lnpopdens+agelt18+age1834+age65over
   +imdens+perwhite+perasian+perblack+perlatin+edhigher+
   edhs+edlhs+unempmale+unempfem+perpov_q+divorce2per+
   samesexper+evprot10+hhsize+occprofman+medinc+hsdrop1619,
   minlambda1 = 2, maxlambda1 = 1000, fold = 10, standard-
   ize = TRUE)
bestfit$lambda
```

Here, after performing 21 iterations, the program settles on an optimal value for lambda1: 27.14285. As stated before, at this low value the penalty parameter will only marginally affect our regression coefficients, though it does increase predictive accuracy slightly.

SUMMARY

The LASSO is a powerful variable selection tool, used to find a subset of available predictors that together have strong predictive power. Data miners use it to optimize both model simplicity (parsimony) and predictive power. Unfortunately, the LASSO is not yet available in some of the more user-friendly data mining packages, so we demonstrated the technique using the free R language.

In the example, we used the LASSO to predict the percentage of the vote in U.S. counties that went to President Obama in the 2012 election. The program identified higher population density, lower percentage of the population under 18, lower percentage white, higher percentage of blacks and percentage with college degrees, and higher male unemployment rate as core predictors of Obama's vote share at the county level.

VIF REGRESSION

VIF regression is another very recently developed data mining tool for simplifying a model by selecting variables (feature selection). Developed in 2011 by Lin, Foster, and Ungar, VIF regression was designed specifically for use with *very* large datasets, and especially those which are very wide (large numbers of variables). It was developed as an alternative to stepwise regression and best-subset regression, both of which are computationally intensive and therefore tend to run very slowly. Other feature selection methods, like generalized path seeker (GPS), run much more quickly but pay a price in reduced predictive accuracy (Lin, Foster, and Ungar 2011). VIF regression is designed to speed up stepwise regression without a substantial trade-off in terms of accuracy.

VIF is a complicated, multistage algorithm, which blends together a number of preexisting techniques (like forward stagewise regression and alpha-investing rules) and adds its own unique element. It is a variation on forward stepwise regression (because with a large number of candidate features, backwards elimination is highly inefficient). We will run through what exactly this algorithm does, and then show how it can be implemented using R.

The main benefit of VIF regression is in reducing the amount of computation that occurs when we run a stepwise regression. But why does stepwise regression require so much mathematical calculation? There are two reasons. First, at each iteration or step in model building, it considers *each candidate variable* for inclusion. This means that at every step, stepwise considers many variables, and since it performs a large number of steps, it does this many times. By contrast, VIF regression runs through the candidate variables only once. Secondly, at each step, stepwise generates estimates for *each* parameter. And it runs not just one regression, but as many regressions as there are candidate variables at each stage. VIF regression gets around this second problem by employing what is called *forward stagewise* regression.

VIF regression starts with a null model—that is, a model containing only an intercept—and calculates residuals from this model. It then selects the first variable in the prespecified list of predictors (the order of which is somewhat important here), and regresses these residuals on this variable. If the variable meets certain criteria for inclusion, it is entered into the model, and new residuals are calculated. If not, the algorithm moves on to the next variable.

At each step, then, only the residuals from the previous stage are regressed on the new candidate variable. This forward stagewise regression permits a substantial reduction in computation. Rather than running a full regression and calculating all the parameter estimates, VIF regression in essence calculates only a series of bivariate correlations.

However, there is a difficulty with this procedure. The *t*-ratios evaluating these bivariate correlations are likely to be biased against variables for which multicollinearity with the variables already selected for the model is substantial. As a result, a "naive" forward stagewise regression algorithm is biased toward selecting variables which are uncorrelated with the variables already in the model rather than selecting what might be the most predictive variable. Some correction must be made to remove this bias.

The correction is what gives VIF regression its name. VIF regression adjusts the *t*-ratios to account for multicollinearity by making use of the "variance inflation factor" of each variable as it comes under consideration. But since the VIF is calculated by regressing a new variable on all the variables already in the model, this presents a conundrum. Performing these regressions would eliminate the savings in computation that was the purpose of using forward stagewise regression in the first place. The solution is to avoid calculating the VIFs using the full dataset. Instead, a small subset of cases is randomly sampled, and VIFs are estimated from this smaller amount of data.

Finally, an allowance is made for an inflation of the probability of type I error over multiple hypothesis tests. Simply put, when we conduct a test for statistical significance, we allow for the possibility of erroneously rejecting a true null hypothesis (type I error), and we choose the probability that this will happen by setting our alpha. But as we conduct multiple hypothesis tests, the likelihood of a rare event increases, so simply applying the same alpha level to each test is not appropriate.

Bonferroni corrections for multiple hypothesis tests simply divide alpha (usually $p < .05$) by n, the number of tests to be performed (i.e. the number of candidate variables). But this correction is problematic if the number of tests is very large, and it elevates the probability of type II error by setting the effective alpha for entry very, very low indeed.

Instead, VIF regression employs a procedure, called an *alpha-investing rule*, which strikes a middle ground between unconstrained multiple-hypothesis testing (which produces a lot of type I error) and the application of a Bonferroni rule (which tends to knock out potentially significant predictors; see Foster and Stine 2008). The idea is that we begin with a certain "wealth," or allowance for type I error (say 0.05 or 0.10). We then perform a hypothesis test. If the null hypothesis is rejected, we *add* to our wealth. And if we fail to reject it, we subtract from our wealth. After all the wealth is exhausted, no more hypothesis tests are permitted. The critical level for inclusion, meanwhile, changes as a function of the current wealth and the number of iterations since the last null-hypothesis rejection. This procedure has been shown to effectively control the likelihood of false rejections of the null hypothesis (Foster and Stine 2008).

So, VIF regression runs through each candidate predictor only once, and admits it only if it passes a fairly high bar for inclusion. But doesn't this mean that the algorithm could "miss" important predictors? The algorithm's creators assure us that this is not the case. If highly predictive variables are uncorrelated with residuals, they will certainly be selected. If a new variable is highly correlated with residuals, then the alpha-investment rule guarantees that the model as a whole will be predictive, though it doesn't guarantee that any one variable will enter the model per se. Users of this technique are advised to prioritize the most important variables by listing them first. The program's authors claim that, for prediction purposes, it doesn't matter whether one or another highly correlated variable enters the model. "If this importance can be washed out or masked by other variables," they write, "then for prediction purposes, there is no difference between the variable and its surrogates, thus neither of them can be claimed 'true'" (Lin, Foster, and Ungar 2011, 239). "Globally important variables" will not be missed, and high-signal predictors will not be missed as long as there aren't other variables which are substantially correlated with them.

The VIF regression algorithm has been shown to be substantially faster than competing algorithms (GPS comes closest), and good at controlling the marginal false-discovery rate (though not as good as stepwise, the forward-backward greedy algorithm [FoBa] or GPS). It has better out-of-sample performance (more predictive accuracy) than GPS and the LASSO, and as good as FoBa and stepwise.

RUNNING VIF REGRESSION: AN EXAMPLE USING R

To our knowledge, the only way to implement VIF regression is through the R package VIF, written and maintained by Dongyu Lin (2011), one of the developers of the method. We perform an example of VIF regression using our county-level 2012 election dataset. As with our example demonstrating LASSO, we will be modeling Obama's share of the vote at the county level.

We first download the package from http://cran.r-project.org, and install it in working memory:

```
install.packages("VIF")
library(VIF)
```

Next, we turn a set of predictors into a matrix, because VIF works better if it deals with the *X*'s as a single object. R's `cbind` command will suffice for this purpose. Note here that all of the variables have the letter *z* as a prefix. We do this to signify the fact that we have *z*-score standardized all of the predictors we are using. This must be done to put all variables on the same scale so that their relative contribution to explaining variance can be properly assessed. This is generally necessary in any feature selector in order to not bias the algorithm in its selection. However, while a package like "penalized" standardizes the variables for you as an option, with VIF you must have already done it yourself, ahead of time.

```
X<-cbind(zlnpop, zlnpopdens, zagelt18, zage1834, zage3564,
  zagegt65, zperwhite, zperblack, zperamind, zperasian,
  zperpacisl, zperother, zpermultirace, zperlatin, zhh-
  size, zlthsed, zhsed, zsomecol, zbached, mastersed,
  zprofed, zdoced, zmalunemp, zmedinc, zperpov, zimdens,
  zprofmanocc, zdivorce2, zsamesex, zhigheredpop)
```

Now that we have gathered all of our candidate predictors into a matrix, we are ready to perform VIF. We run it with the call:

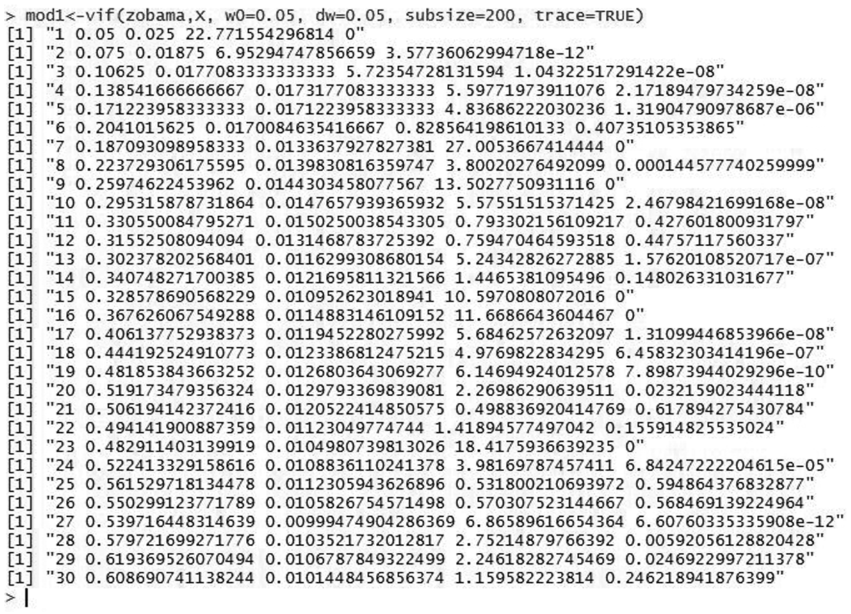
```
mod1<-vif(zobama,X, w0 = 0.05, dw = 0.05, subsize = 200,
  trace = TRUE)
```

This generates an object called "`mod1`," into which the results of the VIF variable selection process will be stored. The `w0` option tells the program the initial wealth we want the model to spend, and `dw` tells it the change in wealth if another variable is added to the model. For more conservative model that selects fewer variables, we would set these values lower. Conversely, setting initial wealth or change in wealth higher will result in the inclusion of more variables. `Subsize` tells the program the size of the random subsample on which to calculate the variance inflation factor of each variable under consideration. Finally, `trace = TRUE` lets us see what goes on as VIF runs through the set of 30 variables we have offered it for evaluation. Doing so generates the output shown in figure 6.5.

We can see that there are 30 rows, one for each of the 30 predictor variables, and five columns. The first number in a column (after the [1] symbol) simply tells us which variable the program is evaluating. The other columns tell us:

1. the current wealth (before the current variable is evaluated)
2. the current test level (which, remember, alters with each new variable, depending on whether prior variables made it into the model or not)
3. the *t*-statistic for the variable under evaluation
4. the *p*-value for this *t*-test

What does all this mean? Well, consider what happens in the first two lines. In line 1, we have the wealth that we chose to begin with: 0.05. To be included in the model, the variable must be significant at $\alpha = .025$ (or the current wealth divided by 2). We see that the result of the *t*-test is 22.77, which is significant far below $p < .001$ (rounded here to 0). This means that the first variable we offered to VIF, zlnpop (the *z*-standardization of the natural log of population density), explained sufficient variance to be included in the model. In line 2, we see the result: the model wealth has increased, while



```
> mod1<-vif(zobama,X, w0=0.05, dw=0.05, subsize=200, trace=TRUE)
[1] "1 0.05 0.025 22.771554296814 0"
[1] "2 0.075 0.01875 6.95294747856659 3.57736062994718e-12"
[1] "3 0.10625 0.0177083333333333 5.72354728131594 1.04322517291422e-08"
[1] "4 0.138541666666667 0.0173177083333333 5.59771973911076 2.17189479734259e-08"
[1] "5 0.171223958333333 0.0171223958333333 4.83686222030236 1.31904790978687e-06"
[1] "6 0.2041015625 0.0170084635416667 0.828564198610133 0.40735105353865"
[1] "7 0.187093098958333 0.0133637927827381 27.0053667414444 0"
[1] "8 0.223729306175595 0.0139830816359747 3.80020276492099 0.000144577740259999"
[1] "9 0.25974622453962 0.0144303458077567 13.5027750931116 0"
[1] "10 0.295315878731864 0.0147657939365932 5.57551515371425 2.46798421699168e-08"
[1] "11 0.330550084795271 0.0150250038543305 0.793302156109217 0.427601800931797"
[1] "12 0.31552508094094 0.0131468783725392 0.759470464593518 0.44757117560337"
[1] "13 0.302378202568401 0.0116299308680154 5.24342826272885 1.57620108520717e-07"
[1] "14 0.340748271700385 0.0121695811321566 1.4465381095496 0.148026331031677"
[1] "15 0.328578690568229 0.010952623018941 10.5970808072016 0"
[1] "16 0.367626067549288 0.0114883146109152 11.6686643604467 0"
[1] "17 0.406137752938373 0.0119452280275992 5.68462572632097 1.31099446853966e-08"
[1] "18 0.444192524910773 0.0123386812475215 4.9769822834295 6.45832303414196e-07"
[1] "19 0.481853843663252 0.0126803643069277 6.14694924012578 7.89873944029296e-10"
[1] "20 0.519173479356324 0.0129793369839081 2.26986290639511 0.0232159023444118"
[1] "21 0.506194142372416 0.0120522414850575 0.498836920414769 0.617894275430784"
[1] "22 0.494141900887359 0.01123049774744 1.41894577497042 0.155914825535024"
[1] "23 0.482911403139919 0.0104980739813026 18.4175936639235 0"
[1] "24 0.52241332915861 6 0.0108836110241378 3.98169787457411 6.84247222204615e-05"
[1] "25 0.561529718134478 0.0112305943626896 0.531800210693972 0.594864376832877"
[1] "26 0.550299123771789 0.0105826754571498 0.570307523144667 0.568469139224964"
[1] "27 0.539716448314639 0.00999474904286369 6.86589616654364 6.60760335335908e-12"
[1] "28 0.579721699271776 0.0103521732012817 2.75214879766392 0.00592056128820428"
[1] "29 0.619369526070494 0.0106787849322499 2.24618282745469 0.0246922997211378"
[1] "30 0.608690741138244 0.0101448456856374 1.15958222381 4 0.246218941876399"
> |
```

FIGURE 6.5

Output displaying variable selection from VIF regression in R.

critical value for inclusion of the *next* variable is set lower (at $\alpha = .018$). Once again, the t-statistic for this second variable is quite high (6.953), and the variable is admitted into the model.

We can see what happens, conversely, when a variable fails to make it into the model by looking at what happens before and after variable 6. Note that the model wealth rises for each variable from 1 through 6. This is, remember, the wealth of the model *before* the new variable is auditioned. Variable 6 does not make it into the model (which we can tell by looking at the t-value, 0.829, and the p-value, 0.407). Note that for variable 7 the wealth dips slightly (from 0.204 to 0.187).

After we have run through our variables, what do we get in terms of model fit? Actually, the VIF routine does not fit the model for you. Instead, it tells you which variables you ought to include and which you should reject. It is, then, a pure *feature selector.*

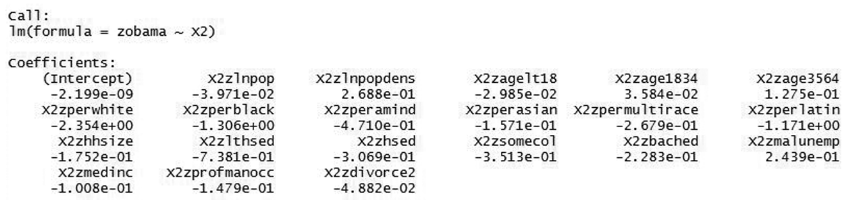
To see the selected variables, call:

```
mod1$select
```

R returns:

```
[1] 1 2 3 4 5 7 8 9 10 13 15 16 17 18 19 23 24 27 28
```

This tells us the ID numbers of the variables—19 in total—which were selected by the model. We see, by examining the list, that many variables are missing: variables 6, 11, 12,



```
Call:
lm(formula = zobama ~ X2)

Coefficients:
    (Intercept)        X2zlnpop     X2zlnpopdens        X2zagelt18        X2zage1834        X2zage3564
     -2.199e-09      -3.971e-02        2.688e-01        -2.985e-02         3.584e-02         1.275e-01
    X2zperwhite     X2zperblack      X2zperamind       X2zperasian  X2zpermultirace       X2zperlatin
     -2.354e+00      -1.306e+00       -4.710e-01        -1.571e-01        -2.679e-01        -1.171e+00
      X2zhhsize       X2zlthsed          X2zhsed       X2zsomecol         X2zbached       X2zmalunemp
     -1.752e-01      -7.381e-01       -3.069e-01        -3.513e-01        -2.283e-01         2.439e-01
      X2zmedinc   X2zprofmanocc     X2zdivorce2
     -1.008e-01      -1.479e-01       -4.882e-02
```

FIGURE 6.6
Output from VIF regression in R.

14, 20, 21, 22, 25, 26, 29, and 30. It is important to remember that the selection of variables by VIF regression depends, to some degree, on the order in which you list them. VIF only tries each variable once, and is simply attempting to maximize explanatory power without overfitting. So if we are trying to ensure that the algorithm picks the "true" variables, it is a good idea to run it a number of times, switching up the order of the variables each time.

If we reduce the value of the `dw` parameter, we reduce the number of variables admitted to the model. That is because the critical value for inclusion in the model depends on model wealth. Adding less wealth upon the rejection of a null hypothesis leads to lower critical values for inclusion, and thus for fewer features selected. When we set `dw` to 0.05, VIF selects 19 variables. But this parameter does not lead quickly to a more parsimonious model. When we drop `dw` to 0.01, 0.001, and 0.0001, we select 18, 18, and 17 variables, respectively. Another option is to reduce the initial wealth of the model. But once again, one must, with this selection of variables, set `w0` quite low before the model becomes substantially smaller.

After settling on the variables for inclusion, we simply run a linear model with only those selected variables. We manually construct a matrix that includes only the subset of selected variables, and then run a linear model on this subset. R regression output appears in figure 6.6.

```
X2<-cbind(zlnpop, zlnpopdens, zagelt18, zage1834, zage3564,
  zperwhite, zperblack, zperamind, zperasian, zpermulti-
  race, zhhsize, zlthsed, zhsed, zsomecol, zbached,
  zmalunemp, zmedinc, zprofmanocc, zdivorce2)
mod2<-lm(zobama~X2)
```

VIF has selected the variables which we have already seen to be important in the prediction of county-level Obama vote share: population density, the percentage of the population that is non-Hispanic white, the percentage of the population that is black, the proportion of young adults in the population, and the male unemployment rate. Thus, this highly efficient algorithm produces results which largely agree with the results of models we have seen before.

As the expansion of the data mining universe proceeds apace, forever churning out new algorithms, researchers have developed yet another technique which allegedly improves upon VIF regression. This latest method, *robust VIF regression*, addresses the tendency of “standard” VIF regression to be sensitive to the presence of outliers in the data (Dupuis and Victoria-Feser 2013). VIF regression is an interesting way of selecting variables in an efficient manner to maximize predictive accuracy.