16
Building predictive models

Building predictive models

Suppose you have a house in Saratoga, NY that you're about to put up for sale. It's a 1900 square-foot house on a 0.7-acre lot. It has 3 bedrooms, 2.5 bathrooms,[1] 1 fireplace, gas heating, and central air conditioning. The house was built 16 years ago. How much would you expect it to sell for?

[1] A half-bathroom has a toilet but no bath or shower.

Although we've been focusing on only a few variables of interest so far, our house-price data set actually has information on all these variables, and a few more besides. A great way to assess the value of the house is to use the available data to fit a multiple regression model for its price, given its features. We can then use this model to make a best guess for the price of a house with some particular combination of features—and, optionally, to form a *prediction interval* that quantifies the uncertainty of our guess.

We refer to this as the process of *building a predictive model.* Although we will still use multiple regression, the goal here is slightly different than in the previous examples we've look at through the lens of multiple regression. Here, we don't care so much about isolating and interpreting one particular partial relationship (like that between fireplaces and price). Instead, we just want the most accurate predictions possible. Interpreting coefficients and worrying about confounders is of secondary concern.

The key principle in building predictive models is *Occam's razor*, which is the broader philosophical idea that models should be only as complex as they need to be in order to explain reality well. The principle is named after a medieval English theologian called Willam of Occam. Since he wrote in Latin, he put it like this: *Frustra fit per plura quod potest fieri per pauciora* ("It is futile to do with more things that which can be done with fewer.") A more modern formulation of Occam's razor might be the KISS rule: keep it simple, stupid.

In regression modeling, this principle is especially relevant for *variable selection*—that is, deciding which possible predictor variables to add to a model, and which to leave out. In this context, Occam's razor is about finding the right set of variables to include so that we fit the data, without overfitting the data. Another way of saying this is that we want to find the patterns in the data, without memorizing the noise.

In this chapter, we'll consider two main questions:

(1) How can we measure the predictive power of a model?

(2) How can we find a model with good predictive power?

Measuring generalization error

To understand how we measure the predictive power of a regression model, we first need a bit of notation. Specifically, let's say that we have estimated a multiple regression model with p predictors $(x_1, x_2, \ldots, x_p)$ to some data, giving us coefficients $(\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p)$. Now we encounter a new case, not in our original data set. We'll let $x^\star = (x_1^\star, x_2^\star, \ldots, x_p^\star)$ be the predictor variables for this new case, and $y^\star$ denote the corresponding response. We will use the fitted regression model, together with $x^\star$, to make a prediction for $y^\star$:

$$\hat{y}^\star = \hat{\beta}_0 + \sum_{j=1}^{p} \hat{\beta}_j x_j^\star \, .$$

Our goal is to make the *generalization error*—that is, the difference between $y^\star$ and $\hat{y}^\star$—as small as possible, on average.

A natural way to measure the generalization error of a regression model is using a quantity called the *mean-squared predictive error*, or MSPE. The mean-squared predictive error is a property of a fitted model, not an individual data point. It summarizes the magnitude of the errors we typically make when we use the model to make predictions $\hat{y}^\star$ on new data:

$$\text{MSPE} = \text{Average value of } (y^\star - \hat{y}^\star)^2 \text{ when sampling new data points} \, .$$

Here a "new" data point means one that hasn't been used to fit the model. You'll notice that, in calculating MSPE, we square the prediction error $y^\star - \hat{y}^\star$ so that both positive and negative errors count equally.

Low mean-squared predictive error means that $y^\star - \hat{y}^\star$ tends to be close to zero when we sample new data points. This gives us a simple principle for building a predictive model: find the model (i.e. the set of variables to include) with the lowest mean-squared predictive error.

Estimating the the mean-squared predictive error

Conceptually, the simplest way to estimate the mean-squared predictive error of a regression model is to actually collect new data and calculate the average predictive error made by our model. Specifically, suppose that, after having fit our model in the first place, we go out there and collect $n^\star$ brand new data points, with responses $y_i^\star$ and predictors $(x_{i1}^\star, \ldots, x_{ip}^\star)$. We can then estimate the mean-squared predictive error of our model in two simple steps:

1. Form the prediction for each new data point:

$$\hat{y}_i{}^\star = \hat{\beta}_0 + \sum_{j=1}^{p} \hat{\beta}_j x_{ij}^\star .$$

2. Calculate the average squared error of your predictions:

$$\widehat{\text{MSPE}}_{\text{out}} = \frac{1}{n^\star} \sum_{i=1}^{n^\star} (y_i^\star - \hat{y}_i^\star)^2 .$$

   Notice that we put a hat on MSPE, because the expression on the right-hand side is merely an *estimate* of the true mean-squared predictive error, calculated using a specific sample of new data points. (Calculating the *true* MSPE would require us, in principle, to average over all possible samples of new data points, which is obviously impractical.) We also use the subscript "out" to indicate that it is an *out-of-sample* measure—that is, calculated on new data, that falls outside of our original sample.

Conventionally, we report the square root of $\widehat{\text{MSPE}}_{\text{out}}$ (which is called *root mean-squared predictive error*, or RMSPE), because this has the same units as the original y variable. You can think of the RMSPE as the standard deviation of future forecasting errors made by your model.

Assuming your new sample size $n^\star$ isn't too small, these two steps are a nearly foolproof way to estimate the mean-squared predictive error of your model. The drawback, however, is obvious: you need a brand new data set, above and beyond the original data set that you used to fit the model in the first place. This new data set might be expensive or impractical to collect.

Thus we're usually left in the position of needing to estimate the mean-squared predictive error of a model, without having access to a "new" data set. For this reason, the usual practice is make a *train/test split* of your data: that is, to randomly split your original data set into two subsets, called the *training* and *testing* sets.

- The training set is used only to fit ("train") the model—that is, to estimate the coefficients $(\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p)$.
- The testing set is used only to estimate the mean-squared predictive error of the model. It is not used at all to fit the model. For this reason, the testing set is sometimes referred to as the "hold-out set," since it is held out of the model-fitting process.

From this description, it should be clear that the training set plays the role of the "old" data, while the testing set plays the role of the "new" data.

This gives us a simple three-step procedure for choosing between several candidate models (i.e. different possible sets of variables to include).

(1) Split your data into training and testing sets.

(2) For each candidate model:

   A. Fit the model using the training set.

   B. Calculate $\widehat{\text{MSPE}}_{\text{out}}$ for that model using the testing set.

(3) Choose the model with the lowest value of $\widehat{\text{MSPE}}_{\text{out}}$.

Choosing the training and testing sets. A key principle here is that you must *randomly* split your data into a training set and testing set. Splitting your data nonrandomly—for example, taking the first 800 rows of your data as a training set, and the last 200 rows as a testing set—may mean that your training and testing sets are systematically different from one another. If this happens, your estimate of the mean-squared prediction error can be way off.

How much of the data should you reserve for the testing set? There are no hard-and-fast rules here. A common rule of thumb is to use about 75% of the data to train the model, and 25% to test it. Thus, for example, if you had 100 data points, you would randomly sample 75 of them to use for model training, and the remaining 25 to estimate the mean-squared predictive error. But other ratios (like 50% training, or 90% training) are common, too.

My general guideline is that the more data I have, the larger the fraction of that data I will use for training the predictive model. Thus with only 100 data points, I might use a 75/25 split between training and testing; but with 10,000 data points, I might use more like a 90/10 split between training and testing. That's because estimating the model itself is generally harder than estimating the mean-squared predictive error.[2] Therefore, as more data accumulates, I like to preferentially allocate more of that data towards the intrinsically harder task of model estimation, rather than MSPE estimation.

[2] By "harder" here, I mean "subject to more sources of statistical error," as opposed to computationally more difficult.

Averaging over different test sets. It's a good idea to average your estimate of the mean-squared predictive error over several different train/test splits of the data set. This reduces the dependence of $\widehat{\text{MSPE}}_{\text{out}}$ on the particular random split into training and testing sets that you happened to choose. One simple way to do this is average your estimate of MSPE over many different random splits of the data set into training and testing sets. Somewhere between 5 and 100 splits is typical, depending on the computational resources available (more is better, to reduce Monte Carlo variability).

Another classic way to estimate MSPE it is to divide your data set into K non-overlapping chunks, called *folds*. You then average your estimate of MPSE over K different testing sets, one corresponding to each fold of the data. This technique is called *cross validation.* A typical choice of K is five, which gives us five-fold cross validation. So when testing on the first fold, you use folds 2-5 to train the model; when testing on fold 2, you use folds 1 and 3-5 to train the model; and so forth.

Can we use the original data to estimate the MSPE?

A reasonable question is: why do even we need a new data set to estimate the mean-squared prediction error? After all, our fitted model has residuals, $e_i = y_i - \hat{y}_i$, which tell us how much our model has "missed" each data point in our sample. Why can't we just use the residual variance, s_e^2, to estimate the MSPE? This approach sounds great on the surface, in that we'd expect the past errors to provide a good guide to the likely magnitude of future errors. Thus you might be tempted to use the *in-sample* estimate of MSPE, denoted

$$\widehat{\text{MSPE}}_{\text{in}} = s_e^2 = \frac{1}{n-p}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \, ,$$

where we recall that p is the number of parameters in the model.

Using $\widehat{\text{MSPE}}_{\text{in}}$ certainly removes the need to collect a new data set. This turns out, however, to be a false economy: $\widehat{\text{MSPE}}_{\text{in}}$ is usually too optimistic as an estimate of a model's generalization error. Practically speaking, this means the following. When we use $\widehat{\text{MSPE}}_{\text{in}}$ to quantify the *in-sample* error of a model, and then we actually go out and take new data to calculate the *out-of-sample* generalization error $\widehat{\text{MSPE}}_{\text{out}}$, we tend to discover that the out-of-sample error is larger—sometimes much larger! This is called overfitting, and it is especially likely to happen when the size of the data set is small, or when the model we're fitting is very complex (i.e. has lots of parameters).

An example

Let's see these ideas in practice, by comparing three predictive models for house prices in Saratoga, New York. Our models will draw from the following set of variables:

- lot size, in acres
- age of house, in years
- living area of house, in square feet
- percentage of residents in neighborhood with college degree
- number of bedrooms
- number of bathrooms
- number of total rooms
- number of fireplaces
- heating system type (hot air, hot water, electric)
- fuel system type (gas, fuel oil, electric)
- central air conditioning (yes or no)

We'll consider three possible models for price constructed from these 11 predictors.

Small model: price versus lot size, bedrooms, and bathrooms (4 total parameters, including the intercept).

Medium model: price versus all variables above, main effects only (14 total parameters, including the dummy variables).

	In-sample RMSPE	Out-of-sample RMSPE	Difference
Small model: underfit	\$76,144	\$76,229	\$85
Medium model: good fit	\$65,315	\$65,719	\$403
Big model: overfit	\$61,817	\$71,426	\$9,609

Table 16.1: In-sample versus out-of-sample estimates of the root mean-squared predictive error for three models of house prices in Saratoga, NY. The "difference" column shows the difference between the in-sample and out-of-sample estimates. The big model has a very large difference (over \$9,000), indicating that the in-sample estimate is way too optimistic, and that the model is probably overfit to the data.

Big model: price versus all variables listed above, together with all pairwise interactions between these variables (90 total parameters, include dummy variables and interactions).

Table 16.1 shows both $\widehat{\text{MSPE}}_{\text{in}}$ and $\widehat{\text{MSPE}}_{\text{out}}$ for these three models. To calculate $\widehat{\text{MSPE}}_{\text{out}}$, we used 80% of the data as a training set, and the remaining 20% as a test set, and we averaged over 100 different random train/test splits of the data. The final column, labeled "difference," shows the difference between the in-sample and out-of-sample estimates of prediction error.

There are a few observations to take away from Table 16.1. The first is that the big model (with all the main effects and interactions) has the lowest in-sample error. With a residual standard deviation of \$61,817, it seems nearly \$3,500 more accurate than the medium model, which is next best. This is a special case of a very general phenomenon: a more complex model will always fit the data better, because it has more degrees of freedom to play with.

However, the *out-of-sample* measure of predictive error tells a different story. Here, the medium-sized model is clearly the winner. Its predictions on new data are off by about \$65,719, on average, which is nearly \$6,000 better than the big model.

Finally, notice how severely degraded the predictions of the big model become when moving from old (in-sample) data to new (out-of-sample) data: about \$9,600 worse, on average. This kind of degradation is a telltale sign of overfitting. The medium model suffers only a mild degradation in performance on new data, while the small model suffers hardly any degradation at all—although it's still not competitive on the out-of-sample measure, because it wasn't that good to begin with. This is also a special case of a more general phenomenon: *some* degradation in predictive performance on out-of-sample versus in-sample data is inevitable, but simpler models tend to degrade a lot less.

Figure 16.1 demonstrates this point visually. Starting from a very simple model of price (using only lot size as a predictor), we've added one variable or interaction at a time[3] from the list on

[3] To be specific here, at each stage we added the single variable or interaction that most improved the fit of the model. See the next section on stepwise selection.

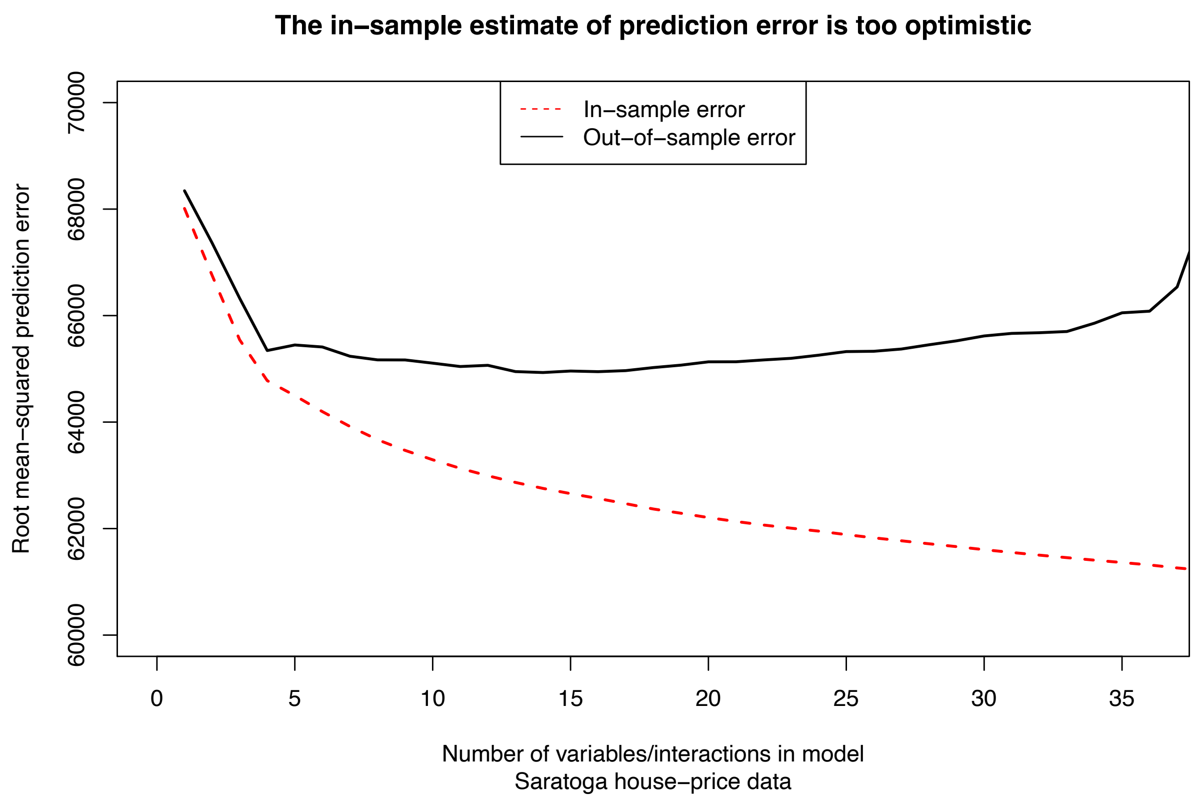

Figure 16.1: Starting from a small pricing model with just lot size as a predictor, we've added one variable or interaction at a time from the list on page 174. The red line shows the in-sample estimate of error, while the black line shows the out-of-sample estimate. After we add about 15 variables and interactions, the out-of-sample error starts to creep back up. Clearly the in-sample estimate is too optimistic, especially as the model gets more complex.

page 174. For each new variable or interaction, we recalculated both the in-sample ($\widehat{\text{MSPE}}_{\text{in}}$) and out-of-sample ($\widehat{\text{MSPE}}_{\text{out}}$) estimates of the generalization error. As we add variables, the out-of-sample error initially gets smaller, reflecting a better fitting model that still generalizes well to new data. But after 15 or 20 variables, eventually the out-of-sample error starts creeping back up, due to overfitting. The in-sample estimate of error, however, keeps going down, falling even further out of line with the real out-of-sample error as we add more variables to the model.

In summary, you should remember the basic mantra of predictive model building: out-of-sample error is larger than in-sample error, especially for bigger models. If you care about minimizing out-of-sample error, you should always use an out-of-sample estimate of a model's MSPE, to make sure that you're not overfitting the original data. Our goal here should be obvious: to find the "turning point" in Figure 16.1, and to stop adding variables before we start overfitting.

Iterative model building via stepwise selection

NOW that we know how to measure generalization error of a model, we're ready to introduce the overall steps in the process of building and using a predictive model from a set of candidate variables x_1, x_2, etc. We sometimes use the term *scope* to refer to this set of candidate variables.

The seemingly obvious approach is to fit all possible models under consideration to a training set, and to measure the generalization error of each one on a testing set. If you have only a few variables, this will work fine. For example, with only 2 variables, there are only $2^2 = 4$ possible models to consider: the first variable in, the second variable in, both variables in, or both variables out. You can fit and test those four models in no time. This is called *exhaustive enumeration*.

However, if there are lots of variables, exhaustive enumeration of all the models becomes a lot harder to do, for the simple fact that it's too exhausting—there are too many models to consider. For example, suppose we have 10 possible variables, each of which we could put in or leave out of the model. Then there are $2^{10} = 1024$ possible models to consider, since each variable could be in or out in any combination. That's painful enough. But if there are 100 possible variables, there are 2^{100} possible models to consider. That's *1 nonillion* models—about 10^{30}, or a thousand billion billion billion. This number is larger than the number of atoms in a human body.

You will quite obviously never be able to fit all these countless billions of models, much less compare their generalization errors on a testing set, even with the most powerful computer on earth. Moreover, that's for just 100 candidate variables *with main effects only*. Ideally, we'd like the capacity to build a model using many more candidate variables than that, or to include the possibility of interactions among the variables.

Thus a more practical approach to model-building is *iterative*: that is, to start somewhere reasonable, and to make small changes to the model, one variable at a time. Model-building in this iterative way is really a three-step process:

(1) Choose a baseline model, consisting of initial set of predictor variables to include in the model, including appropriate transformations, polynomial terms and interactions. Exploratory

data analysis (i.e. plotting your data) will generally help you get started here, in that it will reveal obvious relationships in the data. Then fit the model for y versus these initial predictors.

(2) Check the model. If necessary, change what variables are included, what transformations are used, etc.:

   (a) Are the assumptions of the model met? This is generally addressed using residual plots, of the kind shown in Figures 14.7 and **??**. This allows you to assess whether the response varies linearly with the predictors, whether there are any drastic outliers, etc.

   (b) Are we missing any important variables or interactions? This is generally addressed by *adding* candidate variables or interactions to the model from step (1), to see how much each one improves the generalization error (MSPE).

   (c) Are there signs that the model might be overfitting the data? This is generally addressed by *deleting* variables or interactions that are already in the model, to see if doing so actually improves the model's generalization error.

   You may need to iterate these three questions a few times, going through many rounds of adding or deleting variables, before you're satisfied with your final model. Remember that the best way to measure generalization error is using an out-of-sample measure, like $\widehat{\text{MSPE}}_{\text{out}}$ derived from a train/test split of the data.

   Once you're happy with the model itself, then you can. . . .

(3) Use your fitted model to form predictions (and optionally, prediction intervals) for your new data points.

Can this process be automated?

In this three-step process, step 1 (start somewhere reasonable) and step 3 (use the final model) are usually pretty easy. The part where you'll spend the vast majority of your time and effort is step 2, when you consider many different possible variables to add or delete to the current model, and check how much they improve or degrade the generalization error of that model.

This is a lot easier than considering all possible combinations of variables in or out. But with lots of candidate variables, even this iterative process can get super tedious. A natural question is, can it be automated?

The answer is: sort of. We can easily write a computer program that will automatically check for iterative improvements to some baseline ("working") model, using an algorithm called *stepwise selection*:

(1) From among a candidate set of variables (the scope), check all possible one-variable additions or deletions from the working model;

(2) Choose the single addition or deletion that yields the best improvement to the model's generalization error. This becomes the new "working model."

(3) Iteratively repeat steps (1) and (2) until no further improvement to the model is possible.

The algorithm terminates when it cannot find any one-variable additions or deletions that will improve the generalization error of the working model.

Some caveats. Stepwise selection tends to work tolerably well in practice. But it's far from perfect, and there are some important caveats. Here are three; the first one is minor, but the second two are pretty major.

First, if you run stepwise selection from two different baseline models, you will probably end up with two different final models. This tends not to be a huge deal in practice, however, because the two final models usually have similar mean-squared predictive errors. Remember, when we're using stepwise selection, we don't care too much about *which* combinations of variables we pick, as long as we get good generalization error. Especially if the predictors are correlated with each other, one set of variables might be just as good as another set of similar (correlated) variables.

Second, stepwise selection usually involves some approximation. Specifically, at each step of stepwise selection, we have to compare the generalization errors of many possible models. Most statistical software will perform this comparison *not* by actually calculating $\widehat{\text{MSPE}}_{\text{out}}$ on some test data, but rather using one of several possible heuristic approximations for MSPE. The most common one is called the AIC approximation:[4]

$$\widehat{\text{MSPE}}_{\text{AIC}} = \widehat{\text{MSPE}}_{\text{in}} \left(1 + \frac{p}{n}\right) = s_e^2 \left(1 + \frac{p}{n}\right) ,$$

[4] In case you're curious, AIC stands for "Akaike information criterion." If you find yourself reading about AIC on Wikipedia or somewhere similar, it will look absolutely nothing like the equation I've written here. The connection is via a related idea called "Mallows' C_p statistic," which you can read about here.

where n is the sample size and p is the number of parameters in the model.

The AIC estimate of mean-squared predictive error is not a true out-of-sample estimate, like $\widehat{\text{MSPE}}_{\text{out}}$. Rather, it is like an "inflated" or "penalized" version of the in-sample estimate, $\widehat{\text{MSPE}}_{\text{in}} = s_e^2$, which we know is too optimistic. The inflation factor of $(1 + p/n)$ is always larger than 1, and so $\widehat{\text{MSPE}}_{\text{AIC}}$ is always larger than $\widehat{\text{MSPE}}_{\text{in}}$. But the more parameters p you have relative to data points n, the larger the inflation factor gets. It's important to emphasize that $\widehat{\text{MSPE}}_{\text{AIC}}$ is just an approximation to $\widehat{\text{MSPE}}_{\text{out}}$. It's a better approximation than $\widehat{\text{MSPE}}_{\text{in}}$, but it still relies upon some pretty specific mathematical assumptions that can easily be wrong in practice.

The third and most important caveat is that, when using any kind of automatic variable-selection procedure like stepwise selection, we lose the ability to use our eyes and our brains each step of the way. We can't plot the residuals to check for outliers or violations of the model assumptions, and we can't ensure that the combination of variables visited by the algorithm make any sense, substantively speaking. It's worth keeping in mind that your eyes, your brain, and your computer are your three most powerful tools for statistical reasoning. In stepwise selection, you're taking two of these tools out of the process, for the sake of doing a lot of brute-force calculations very quickly.

None of these caveats are meant to imply that you *shouldn't* use stepwise selection—merely that you shouldn't view the algorithm as having God-like powers for discerning the single best model, or treat it as an excuse to be careless. You should instead proceed cautiously. Always verify that the stepwise-selected model makes sense and doesn't violate any crucial assumptions. It's also a good idea to perform a quick train/test split of your data and compute $\widehat{\text{MSPE}}_{\text{out}}$ for your final model, just as a sanity check, to make sure that you're actually improving the generalization error versus your baseline model.