**3 - Cross-Validation for Estimating Prediction Error**

**3.1 – Introduction to Prediction Error and Estimating It**

In this section we will discuss several strategies for estimating prediction error for predictive models where the response is numeric. These methods are commonly collectively known as cross-validation (CV) procedures. As we have only considered MLR at this point, we consider examples of these cross-validation strategies in the case of MLR models. However these methods of cross-validation can be extended to any of the methods we will examine for predicting a numeric response. They also extend to classification problems as well with changes to the metric we use for measuring the predictive performance.

When the response is numeric we wish to estimate how accurately our models make predictions for future observations. As we have seen, there are different metrics that can be used to measure and compare predictive performance of different models – the mean squared error (MSE), the root mean squared error (RMSE), the mean absolute error (MAE), the mean absolute percentage error (MAPE), etc. All of these can be computed for a given situation as long as the errors are measured when making predictions for observations/cases NOT used in the model development process.

For prediction these measures can be defined as:

Here the observations must **NOT have been used in any way** to develop the estimate fitted model and obtain the prediction values for the response .

Also there certainly other ways to quantify the size of the prediction errors such the median (vs. mean), trimmed means (throw out a certain percentage of the largest errors on each end), quantiles, and other metrics – although these are the main three used.

**3.2 – General Idea Behind Cross-Validation Methods**

As mentioned previously the goal of statistical learning models for a numeric response is to predict the response accurately, possible at the expense of interpretability. The stepwise methods we reviewed in MLR to some extent help identify models that may predict the response well but the criteria (AIC, BIC, Mallow’s , p-values, etc.) used in model selection are NOT considering how accurately our model will predict future values of the response.

In order to measure prediction accuracy we need to assess the ability of the model to predict the response using observations that were not used in the model development process. The reason why this is important is essentially the same reason why we cannot use the unadjusted in the model development process. Every time we add a term to the model the RSS goes down and the R-square goes up. A more complex model will always explain more variation in the response, however that does not mean it is going to predict the response more accurately.

To measure prediction accuracy we use **Cross-Validation**.In cross-validation we essentially divide our available data into disjoint sets of observations. One set of observations, called the ***training*** set, will be used to develop and fit the model. The model developed using the training set will then be used to predict the known response values in the other set, called the ***validation*** set. We use the validation set to choose the model that best predicts the response values in validation set. In order to judge the accuracy of future response predictions using the model selected by using the training and validation sets we may to choose have a third set of observations called the ***test*** set. The test cases are NOT used in the model development process at all, thus the accuracy of the response predictions for the test set cases should give us a reasonable measure of the prediction accuracy of our final model. We will see later in this course that we often times use the validation set to fine tune the model in terms of its predictive abilities, thus in some sense it is using the observations in the validation set for model development purposes (even though they are NOT used to fit model). If we do not have a large dataset we may not have enough observations to create these three sets, in which case we may want only create the training and validation sets. The validation and test sets are also called ***holdback*** sets as they contain observations held out in estimating the model. Most modern regression methods have some form of internal validation built into the algorithm that is used to “select or tune” the model.

There are different approaches one can take in forming training/validation/test sets for the purposes of conducting the cross-validation of a model. We will examine several schemes that are commonly employed below.

**3.3 - Split-Sample Cross-Validation Approaches**

Split-sample approaches simply split our original sample into the disjoint sets defined above. Splitting is usually done randomly, however we may choose to use a stratified sampling to take other factors into account when creating our sets. For example, if one of the variables in our data is the subject’s sex then we may want to make sure our sets are balanced in terms of the distribution of sex. We can also stratify on a numeric variable to make sure the distribution of this variable is roughly same in each set. For example if we are modeling home prices, we may want to make sure each set has a similar mixture of low and high price homes.

**Training/Validation Sets Only**

There is no definitive rule for the percentage of observations assigned to each set. Some common choices would 80/20, 75/25, 70/30, 66.6/33.4, or 60/40 (though if you are willing to use 40% of your data for validation purposes it would be better to use training/validation/test sets.) For multiple regression a rule of thumb that can be used is to assign p% to the validation set, where and   
 the largest number of parameters your model may contain.

Training Set

(100-p)%

Original Dataset

**Split randomly or stratified**

Validation Set  
  
(p%)

**Training/Validation/Test Sets**

Training Set

Original Dataset

Again there is no definitive rule for the percentage of observations assigned to each set, however the most common are 60/20/20 or 70/20/10 with the former being the most common. Note the Test Set is in RED because it is not used in the model development process.

Validation Set

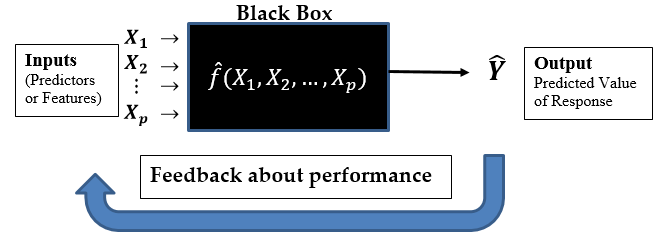
**Split randomly or stratified**

Test Set

The reason the train, validation, and test set approach is used has to do with the diagram below from Section 1. The feedback loop in the diagram below is achieved by using the validation cases provide a measure of predictive performance for the model being fit to the training cases. We will see that many of the algorithms have “tuning” parameters that control the complexity of the estimated function in the “black box”. More complex models will generally fit the training cases better (i.e. have a smaller RSS) but may not necessarily predict response value for the validation cases better than a simpler model. Thus we can fine tune our model fit to the training cases by considering how accurately it predicts the validation cases. Once a “final” model has been chosen using the fitting algorithm under consideration that predicts the validation cases most accurately we can then predict the test cases to obtain a realistic measure of predictive performance for future observations.

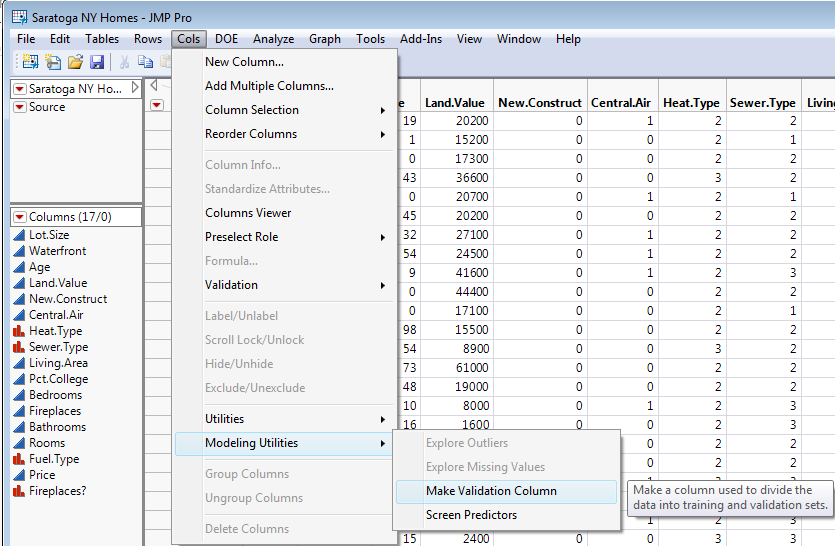
**Training and Validation Sets Test Set**

Final prediction accuracy metrics (RMSEP, MAEP, MAPEP) can be computed for these cases to give a more realistic measure of predictive performance.   
As we will have several methods for estimating the model (MLR, Neural Networks, Random Forests, etc.) in our toolbox to choose from, we can ultimately choose the one with smallest error when predicting the test cases.

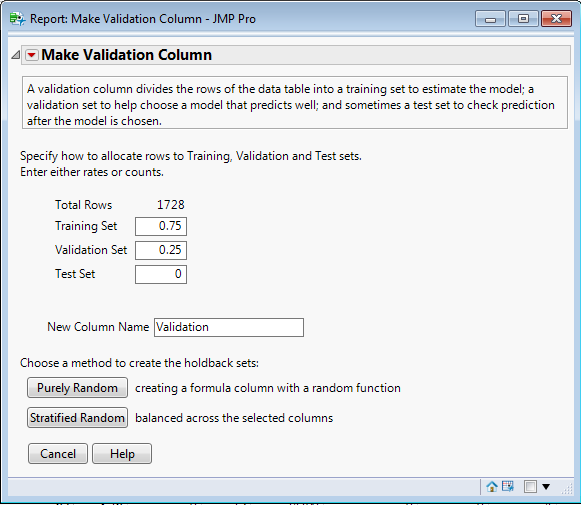


Prediction accuracy for the validation cases is used to provide feedback.

**Aside:**   
JMP makes the process of generating these sets (either training/validation or training/validation/test) very easy by using **Analyze > Predictive Modeling > Make Validation Column** option as shown below.



As you can see we have the option to make only training/validation sets or training/validation/test sets with percentages (proportions) we choose. The default is training/validation sets only with 75% training and 25% validation. We can also make the set assignments randomly or by incorporating stratification variable(s).



However, as we will always have R available to use for free (hopefully), we will now look at how you can create these sets (train/validation or train/validation/test) in R. We saw examples of how to create train/validation sets in the previous section with some of our MLR examples.

**Example 3.1 – Saratoga, NY Home Prices**

In this example we will not focus on model building, but rather the split-sample approach to cross-validation, using either train/validation or train/validation/test sets.

> names(Saratoga)

[1] "Price" "Lot.Size" "Waterfront" "Age" "Land.Value"

[6] "New.Construct" "Central.Air" "Fuel.Type" "Heat.Type" "Sewer.Type"



[11] "Living.Area" "Pct.College" "Bedrooms" "Fireplaces" "Bathrooms"

[16] "Rooms"

> str(Saratoga)

'data.frame': 1728 obs. of 16 variables:

$ Price : int 132500 181115 109000 155000 86060 120000 153000 170000 90000 122900 ...

$ Lot.Size : num 0.09 0.92 0.19 0.41 0.11 0.68 0.4 1.21 0.83 1.94 ...

$ Waterfront : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...

$ Age : int 42 0 133 13 0 31 33 23 36 4 ...

$ Land.Value : int 50000 22300 7300 18700 15000 14000 23300 14600 22200 21200 ...

$ New.Construct: Factor w/ 2 levels "0","1": 1 1 1 1 2 1 1 1 1 1 ...

$ Central.Air : Factor w/ 2 levels "0","1": 1 1 1 1 2 1 1 1 1 1 ...

$ Fuel.Type : Factor w/ 3 levels "2","3","4": 2 1 1 1 1 1 3 3 2 1 ...

$ Heat.Type : Factor w/ 3 levels "2","3","4": 3 2 2 1 1 1 2 1 3 1 ...

$ Sewer.Type : Factor w/ 3 levels "1","2","3": 2 2 3 2 3 2 2 2 2 1 ...

$ Living.Area : int 906 1953 1944 1944 840 1152 2752 1662 1632 1416 ...

$ Pct.College : int 35 51 51 51 51 22 51 35 51 44 ...

$ Bedrooms : int 2 3 4 3 2 4 4 4 3 3 ...

$ Fireplaces : int 1 0 1 1 0 1 1 1 0 0 ...

$ Bathrooms : num 1 2.5 1 1.5 1 1 1.5 1.5 1.5 1.5 ...

$ Rooms : int 5 6 8 5 3 8 8 9 8 6 ...

> dim(Saratoga) 🡨 There are homes available for use in developing a predictive model for price.

[1] 1728 16

> n = dim(Saratoga)[1] 🡨 extract the first of the two dimensions returned by dim.

> n

[1] 1728

> n = nrow(Saratoga) 🡨 this also works

**Creating training and validation sets**As with most things you want to do in R, there is more than one way to skin a cat. I will   
demonstrate a few of them below. The floor function truncates any decimal down to the  
nearest integer value.



> train = sample(1:n,size=floor(n\*.70),replace=F)

> length(train)

[1] 1209

> .70\*n

[1] 1209.6

The training cases can then be referenced by using: DATA[train,]  
The validation cases can then be referenced by using: DATA[-train,]

This is the approach I used in both examples in Chapter 2 where we split our data into these two sets of cases/observations.

**Creating training, validation, and test sets**  
Suppose we wish to split our data into train, validation, and test sets using approximately   
60%-20%-20% of the observations in these sets respectively.

> n = nrow(Saratoga)

> m1 = floor(n\*.60) 🡨 or ceiling(n\*.60)

> m2 = floor(n\*.20) 🡨 or ceiling(n\*.20)

> RO = sample(1:n,size=n,replace=F) 🡨 this command permutes the indices 1 – n.

> train = RO[1:m1]

> valid = RO[(m1+1):(m1+m2+1)]

> test = RO[(m1+m2+2):n]

> length(train)

[1] 1036

> length(valid)

[1] 346

> length(test)

[1] 346

> 1036+346+346

[1] 1728

The training cases can then be referenced by using: DATA[train,]  
The validation cases can then be referenced by using: DATA[valid,]

The test cases can then be referenced by using: DATA[test,]

Before looking at a very simple example using these three sets for the Saratoga, NY home price data we will write a function to compute measures of predictive performance which takes the actual response values and the predicted response values as arguments. It is important to realize that I am a hack R programmer, so my functions are not very elegant or efficient in terms of coding.

PredAcc = function(y,ypred){

RMSEP = sqrt(mean((y-ypred)^2))

MAE = mean(abs(y-ypred))

MAPE = mean(abs(y-ypred)/y)\*100

cat("RMSEP\n")

cat("===============\n")

cat(RMSEP,"\n\n")

cat("MAE\n")

cat("===============\n")

cat(MAE,"\n\n")

cat("MAPE\n")

cat("===============\n")

cat(MAPE,"\n\n")

return(data.frame(RMSEP=RMSEP,MAE=MAE,MAPE=MAPE))

}

We now use a simple model to show how we can use the training, validation, and test sets in development of a MLR regression model for predicting home prices using the Saratoga data.

Note: Here is another command that will create Training, Validation, and Test sets.

CV = sample(c(“Train”,”Valid”,”Test”),size=n,replace=T,prob=c(.60,.20,.20))

home.lm1 = lm(Price~.,data=Saratoga[CV==”Train”,])

ypred = predict(home.lm1,newdata=Saratoga[CV==”Valid”,] etc…

We will first fit a model using the response and all of the numeric predictors in their original scale as well as all of the factor terms created from the nominal predictors using the training data.

> home.lm1 = lm(Price~.,data=Saratoga[train,])

> summary(home.lm1)

Call:

lm(formula = Price ~ ., data = Saratoga[train, ])

Residuals:

Min 1Q Median 3Q Max

-219094 -34937 -5129 26474 463964

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 8.971e+03 2.586e+04 0.347 0.7288

Lot.Size 7.236e+03 3.018e+03 2.398 0.0167 \*

Waterfront1 8.579e+04 2.100e+04 4.086 4.74e-05 \*\*\*

Age -1.219e+02 7.511e+01 -1.622 0.1050

Land.Value 9.092e-01 7.208e-02 12.614 < 2e-16 \*\*\*

New.Construct1 -5.062e+04 1.114e+04 -4.544 6.19e-06 \*\*\*

Central.Air1 1.191e+04 4.820e+03 2.470 0.0137 \*

Fuel.Type3 -9.904e+03 1.745e+04 -0.567 0.5705

Fuel.Type4 4.197e+03 6.783e+03 0.619 0.5363

Heat.Type3 -7.033e+03 5.698e+03 -1.234 0.2174

Heat.Type4 3.648e+03 1.781e+04 0.205 0.8377

Sewer.Type2 1.532e+04 2.218e+04 0.691 0.4899

Sewer.Type3 2.022e+04 2.215e+04 0.913 0.3614

Living.Area 7.009e+01 6.055e+00 11.576 < 2e-16 \*\*\*

Pct.College -2.910e+02 2.067e+02 -1.408 0.1595

Bedrooms -6.052e+03 3.459e+03 -1.750 0.0805 .

Fireplaces 6.159e+03 4.008e+03 1.537 0.1246

Bathrooms 2.312e+04 4.534e+03 5.098 4.09e-07 \*\*\*

Rooms 2.387e+03 1.299e+03 1.838 0.0663 .

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 61420 on 1017 degrees of freedom

Multiple R-squared: 0.6287, Adjusted R-squared: 0.6222

F-statistic: 95.68 on 18 and 1017 DF, p-value: < 2.2e-16

We then can use our prediction accuracy function above to measure the predictive performance of this model for the validation cases.

> y = Saratoga$Price[valid]

> ypred = predict(home.lm1,newdata=Saratoga[valid,])  
> results = PredAcc(y,ypred)

RMSEP

===============

53383.7

MAE

===============

41583.18

MAPE

===============

38.10673

Because we returned the prediction accuracy measures in the form of a data frame we can assign them to an object called “results”.   
  
> results$RMSEP

[1] 53383.7

> results$MAE

[1] 41583.18

> results$MAPE

[1] 38.10673

> results

RMSEP MAE MAPE

1 53383.7 41583.18 38.10673

We will now construct a simplified model using stepwise model selection to reduce the complexity of our base model. We can then use our prediction accuracy function to compare the two models in terms of their predictive performance on the validation cases.

> home.step = step(home.lm1)

Start: AIC=22863.65

Price ~ Lot.Size + Waterfront + Age + Land.Value + New.Construct +

Central.Air + Fuel.Type + Heat.Type + Sewer.Type + Living.Area +

Pct.College + Bedrooms + Fireplaces + Bathrooms + Rooms

Df Sum of Sq RSS AIC

- Fuel.Type 2 2.7964e+09 3.8394e+12 22860

- Sewer.Type 2 5.9876e+09 3.8426e+12 22861

- Heat.Type 2 5.9985e+09 3.8426e+12 22861

<none> 3.8366e+12 22864

- Pct.College 1 7.4770e+09 3.8441e+12 22864

- Fireplaces 1 8.9111e+09 3.8455e+12 22864

- Age 1 9.9308e+09 3.8465e+12 22864

- Bedrooms 1 1.1549e+10 3.8481e+12 22865

- Rooms 1 1.2745e+10 3.8493e+12 22865

- Lot.Size 1 2.1686e+10 3.8583e+12 22868

- Central.Air 1 2.3021e+10 3.8596e+12 22868

- Waterfront 1 6.2972e+10 3.8996e+12 22879

- New.Construct 1 7.7885e+10 3.9145e+12 22883

- Bathrooms 1 9.8044e+10 3.9346e+12 22888

- Living.Area 1 5.0551e+11 4.3421e+12 22990

- Land.Value 1 6.0022e+11 4.4368e+12 23012

Step: AIC=22860.41

Price ~ Lot.Size + Waterfront + Age + Land.Value + New.Construct +

Central.Air + Heat.Type + Sewer.Type + Living.Area + Pct.College +

Bedrooms + Fireplaces + Bathrooms + Rooms

Df Sum of Sq RSS AIC

- Sewer.Type 2 4.6730e+09 3.8441e+12 22858

- Heat.Type 2 8.9420e+09 3.8483e+12 22859

<none> 3.8394e+12 22860

- Pct.College 1 7.4324e+09 3.8468e+12 22860

- Fireplaces 1 8.5136e+09 3.8479e+12 22861

- Age 1 9.0762e+09 3.8485e+12 22861

- Bedrooms 1 1.2039e+10 3.8514e+12 22862

- Rooms 1 1.3342e+10 3.8527e+12 22862

- Central.Air 1 2.2550e+10 3.8619e+12 22865

- Lot.Size 1 2.3201e+10 3.8626e+12 22865

- Waterfront 1 6.6401e+10 3.9058e+12 22876

- New.Construct 1 7.7217e+10 3.9166e+12 22879

- Bathrooms 1 9.5965e+10 3.9354e+12 22884

- Living.Area 1 5.0854e+11 4.3479e+12 22987

- Land.Value 1 5.9893e+11 4.4383e+12 23009

**ETC…**

> summary(home.step)

Call:

lm(formula = Price ~ Lot.Size + Waterfront + Age + Land.Value +

New.Construct + Central.Air + Living.Area + Bedrooms + Bathrooms +

Rooms, data = Saratoga[train, ])

Residuals:

Min 1Q Median 3Q Max

-218892 -34757 -5124 26623 464177

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 1.007e+04 8.675e+03 1.161 0.24584

Lot.Size 6.562e+03 2.708e+03 2.423 0.01557 \*

Waterfront1 9.041e+04 2.071e+04 4.365 1.40e-05 \*\*\*

Age -1.258e+02 7.072e+01 -1.779 0.07559 .

Land.Value 9.038e-01 6.939e-02 13.025 < 2e-16 \*\*\*

New.Construct1 -4.759e+04 1.088e+04 -4.376 1.33e-05 \*\*\*

Central.Air1 1.445e+04 4.445e+03 3.252 0.00119 \*\*

Living.Area 7.136e+01 5.846e+00 12.208 < 2e-16 \*\*\*

Bedrooms -6.609e+03 3.387e+03 -1.952 0.05127 .

Bathrooms 2.367e+04 4.448e+03 5.321 1.27e-07 \*\*\*

Rooms 2.563e+03 1.292e+03 1.984 0.04751 \*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 61400 on 1025 degrees of freedom

Multiple R-squared: 0.626, Adjusted R-squared: 0.6224

F-statistic: 171.6 on 10 and 1025 DF, p-value: < 2.2e-16

The stepwise reduced models has 8 less terms, thus we are estimating 8 less parameters, resulting in a simpler model than the full model. It should be case that this simpler model has better predictive performance. To see if this is the case, we again use our validation set and measure the predictive accuracy of this simpler model for the validation case response values.

> ypred = predict(home.step,newdata=Saratoga[valid,])

> results.step = PredAcc(y,ypred)

RMSEP

===============

53156.51

MAE

===============

41497.43

MAPE

===============

37.7876

> results

RMSEP MAE MAPE

1 53383.7 41583.18 38.10673

> results.step

RMSEP MAE MAPE

1 53156.51 41497.43 37.7876

As expected, the simpler model has better predictive performance than the larger, more complex, MLR model (though only slightly). At this point we might decide our reduced model is the “best” MLR model we can develop for these data (which I highly doubt it is). Thus we can get a final estimate of the predictive performance of this model for future observations by looking at the prediction accuracy for test cases.

> ypred = predict(home.step,newdata=Saratoga[test,])

> y = Saratoga$Price[test]  
  
> results.test = PredAcc(y,ypred)

RMSEP

===============

54355.93

MAE

===============

39999.6

MAPE

===============

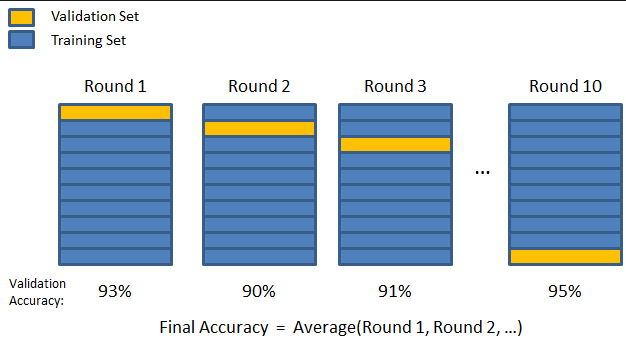
21.53177

These results could now be reported as the expected predictive accuracy of our model as we move forward estimating home selling prices in Saratoga, NY given the home characteristics (i.e. predictor values).

It is important to note that though the model was only fit using the training cases we compared rival models using the prediction accuracy for the validation cases. Thus the validation cases were used in the model development process!

**3.4 – k-Fold Cross-Validation**

Another common cross-validation method used in model development is **k-fold Cross-validation**. In k-fold cross-validation the entire dataset is broken into roughly equal size disjoint sets (k = 5 or 10 typically). Then rounds of model fitting is done where the model is fit using (k-1) of the sets to predict the set left out with of the *k*-sets serving as the validation set. The diagram below illustrates a 10-fold cross-validation ().

10-fold Cross-Validation  


Using this method the model chosen is the one that has the best average or aggregate prediction error over the subsets. Some of the methods we will examine in this course have a built-in k-fold cross-validation in the model fitting process. More specifically the “tuning” parameters in fitting the model are automatically chosen internally using k-fold cross-validation. We still may choose to use a split-sample approach along with the internal k-fold cross-validation however. It is difficult to do k-fold cross-validation for a model method you are considering without writing your own function to do so. There are functions in some packages that will k-fold cross-validation for you however.

Note: k-fold cross-validation with () is essentially equivalent to the training/validation split-sample approach where a 50-50% split is used.

On the following page is code for a function (kfold.MLR) that will perform k-fold cross-validation for any MLR model where the response has NOT been transformed. However, it could be used to compare the predictive performance of rival models where the same response transformation has been used, e.g. for comparing MLR models where the log transformed response is used in each.

**Function for performing k-fold cross-validation of a MLR regression model**

kfold.MLR = function(fit,k=10,data=fit$model) {

sum.sqerr = rep(0,k)

sum.abserr = rep(0,k)

sum.pererr = rep(0,k)

y = fit$model[,1]

x = fit$model[,-1]

n = nrow(data)  
 folds = sample(1:k,nrow(data),replace=T)

for (i in 1:k) {

fit2 <- lm(formula(fit),data=data[folds!=i,])

ypred = predict(fit2,newdata=data[folds==i,])

sum.sqerr[i] = sum((y[folds==i]-ypred)^2)

sum.abserr[i] = sum(abs(y[folds==i]-ypred))  
 sum.pererr[i] = sum(abs(y[folds==i]-ypred)/y[folds==i])

}

cv = return(data.frame(RMSEP=sqrt(sum(sum.sqerr)/n),

MAE=sum(sum.abserr)/n,

MAPE=(sum(sum.pererr)/n)\*100))

}

Comments on the kfold.MLR code:

**I will expect by the end of this course that all of you will be able to write your own cross-validation routines for any modeling method.**

Below the kfold.MLR function is used to compare the full model and the stepwise reduced models for the Saratoga data using MLR with the response and numeric predictors in their original scales.

> home.lm1 = lm(Price~.,data=Saratoga) 🡨 fit using all of the data  
> home.step = step(home.lm1)  
  
> results.full = kfold.MLR(home.lm1)

> results.full

RMSEP MAE MAPE

1 58740.44 41580.13 0.2562079

> results.step = kfold.MLR(home.step)

> results.step

RMSEP MAE MAPE

1 58587.15 41488.02 0.2557484

**3.5 - Leave-Out-One Cross-Validation (LOOCV) and GCV Criterion**

Leave-out-one cross-validation is a quick an easy way to assess prediction accuracy, though DEFINITELY not the best! LOOCV is equivalent to -fold cross-validation where (the number of observations in the data set).

Using the fact the predicted value for when the case is deleted from the MLR model is equal to

Here and .

This is also called the jackknife residual and the sum of these squared jackknife residuals is called the PRESS statistic (Predicted REsidual Sum of Squares), one of the first measures of prediction error.

You can obtain the PRESS statistic for any fitted MLR model in R by running the function PRESS whose code is below. Another way to do this is to run the kfold.MLR function with .

PRESS = function(lm1){

lmi = lm.influence(lm1)

h = lmi$hat

e = resid(lm1)

PRESS = sum((e/(1-h))^2)

RMSEP = sqrt(PRESS/n)

return(data.frame(PRESS=PRESS,RMSEP=RMSEP))

}

> home.lm1 = lm(Price~.,data=Saratoga) 🡨 again fit using all of the full dataset

> home.step = step(home.lm1)  
  
> press.full = PRESS(home.lm1)

> press.step = PRESS(home.step)

> press.full

PRESS RMSEP

1 5.964562e+12 58751.29

> press.step

PRESS RMSEP

1 5.925972e+12 58560.93

Computing the leverage values () is computationally expensive, thus a similar criterion to the PRESS statistic that is computationally much less expensive is the Generalized Cross-Validation statistic.

This definition of the GCV criterion is specific to OLS regression. In matrix notation OLS regression is given by,

where,

in our regression model and the are the observed values of the *jth* term. As before, the OLS estimates of the parameters are found using matrices as:

Notice the predicted values are a linear combination of the observed response values , namely . **This is common to several more advanced modeling strategies we will be examining in the course.**

Many more flexible regression modeling methods have the following property.

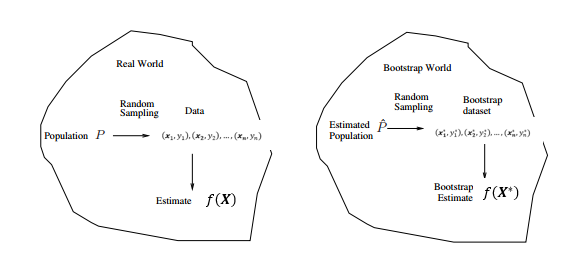
This says that the fitted values from the model are obtained by taking a linear combinations of the observed response values determined by the rows of the matrix **.** For example, regression methods such as ridge and LASSO regression have this form. The subscript represents a “tuning” parameter that controls the complexity of the fitted model. For example in MLR the tuning parameter would essentially be the number of terms/parameters used in the model.

When the fitted values from the model are obtained as the GCV criterion is given by,

Several methods we will examine use the GCV criterion internally to choose the optimal tuning parameter value ().

**3.6 - Bootstrap Estimate of Prediction Error**

The ***bootstrap*** in statistics is a method for approximating the sampling distribution of a statistic by resampling from our observed random sample. To put it simply, a bootstrap sample is a sample of size *n* drawn with replacement from our original sample.   
  
The bootstrap treats the original random sample as the estimated population () and draws repeated samples with replacement from it. For each bootstrap sample we can fit our predictive model.



A bootstrap sample for regression (or classification) problems is illustrated below.

here the are the *p*-dimensional predictor vectors.

where is a random selected observation from the original data drawn with replacement.

We can use the bootstrap sample to calculate any statistic of interest. This process is then repeated a large number of times (B = 500, 1000, 5000, etc.).

For estimating prediction error we fit whatever model we are considering to our bootstrap sample and use it to predict the response value for observations not selected in our bootstrap sample. One can show that about 63.2% of the original observations will represented in the bootstrap sample and about 36.8% of the original observations will not be selected. *Can you show this?* Thus we will almost certainly have some observations that are not represented in our bootstrap sample to serve as a validation set, with the selected observations in our bootstrap sample serving as our training set. For each bootstrap sample we can predict the response for the cases in the validation set (i.e. indices for observations not represented in our bootstrap sample).

**Estimating the prediction error via the .632 Bootstrap**

Again for a numeric response our goal is to estimate the root mean prediction squared error (RMSEP) to compare rival models. Another alternative to the cross-validation methods presented above is to use the .632 bootstrap for estimating the PSE.

The algorithm for the **.632 bootstrap** is given below:

1. First calculate the average squared residual (ASR) from your model

ASR = .

1. Take B bootstrap samples drawn with replacement, i.e. we draw a sample with replacement from the numbers 1 to n and use the cases/observations as our “new data”.
2. Fit the model to each of the B bootstrap samples, computing the from predicting the observations not represented in the bootstrap sample.

Here = average squared residual for prediction in the bootstrap sample, .

1. Compute ASR0 = the average of the bootstrap values
2. Compute the optimism (), .
3. The .632 bootstrap estimate of .

The bootstrap approach has been shown to be better than K-fold cross-validation in many cases.

Here is a function for finding the .632 bootstrap estimate of the RMSEP given a MLR model fit using the lm function.

bootols.cv = function(fit,B=100,data=fit$model) {

yact=fit$fitted.values+fit$residuals

ASR=mean(fit$residuals^2)

AAR=mean(abs(fit$residuals))

APE=mean(abs(fit$residuals)/yact)

boot.sqerr=rep(0,B)

boot.abserr=rep(0,B)

boot.perr=rep(0,B)

y = fit$model[,1]

x = fit$model[,-1]

n = nrow(data)

for (i in 1:B) {

sam=sample(1:n,n,replace=T)

samind=sort(unique(sam))

temp=lm(formula(fit),data=data[sam,])

ypred=predict(temp,newdata=data[-samind,])

boot.sqerr[i]=mean((y[-samind]-ypred)^2)

boot.abserr[i]=mean(abs(y[-samind]-ypred))

boot.perr[i]=mean(abs(y[-samind]-ypred)/y[-samind])

}

ASRo=mean(boot.sqerr)

AARo=mean(boot.abserr)

APEo=mean(boot.perr)

OPsq=.632\*(ASRo-ASR)

OPab=.632\*(AARo-AAR)

OPpe=.632\*(APEo-APE)

RMSEP=sqrt(ASR+OPsq)

MAEP=AAR+OPab

MAPEP=(APE+OPpe)\*100

cat("RMSEP\n")

cat("===============\n")

cat(RMSEP,"\n\n")

cat("MAE\n")

cat("===============\n")

cat(MAEP,"\n\n")

cat("MAPE\n")

cat("===============\n")

cat(MAPEP,"\n\n")

return(data.frame(RMSEP=RMSEP,MAE=MAEP,MAPE=MAPEP))

}

Again we perform cross-validation on the full MLR model for the Saratoga, NY home prices using all numeric variables in the original scales.  
  
> home.lm1 = lm(Price~.,data=Saratoga)  
> results.boot = bootols.cv(home.lm1,B=100)

RMSEP

===============

58873.29

MAE

===============

41630.37

As different bootstrap samples are drawn each time, the results will vary some from run-to-run. As increases however, the variability in the estimates of RMSEP, MAE, and MAPE from run-to-run will decrease.

MAPE

===============

0.2617831

> results.boot = bootols.cv(home.lm1,B=100)

RMSEP

===============

58392.04

MAE

===============

41455.67

MAPE

===============

0.2583147

> results.boot = bootols.cv(home.lm1,B=1000)🡨 increasing the # of bootstrap samples (B = 1000)

RMSEP

===============

58726.64

MAE

===============

41620

MAPE

===============

0.2559745  
  
> results.boot = bootols.cv(home.lm1,B=5000)🡨 increasing the # of bootstrap samples (B = 5000)   
 (takes a quite a while, approximately 1 minute)  
RMSEP

===============

58705.25

MAE

===============

41591.98

MAPE

===============

0.2564237

If response transformations are used the code would need to altered so that the predictive performance measures are computed in the original scale. For example if we log transform the response the function would be:

bootlog.cv = function(fit,B=100,data=fit$model) {

yt=fit$fitted.values+fit$residuals  
 yact = exp(yt)  
 yhat = exp(fit$fitted.values)  
 resids = yact - yhat

ASR=mean(resids^2)

AAR=mean(abs(resids))

APE=mean(abs(resids)/yact)

boot.sqerr=rep(0,B)

boot.abserr=rep(0,B)

boot.perr=rep(0,B)

y = fit$model[,1]

x = fit$model[,-1]

n = nrow(data)

for (i in 1:B) {

sam=sample(1:n,n,replace=T)

samind=sort(unique(sam))

temp=lm(formula(fit),data=data[sam,])

ytp=predict(temp,newdata=data[-samind,])  
 ypred = exp(ytp)

boot.sqerr[i]=mean((exp(y[-samind])-ypred)^2)

boot.abserr[i]=mean(abs(exp(y[-samind])-ypred))

boot.perr[i]=mean(abs(exp(y[-samind])-ypred)/exp(y[-samind]))

}

ASRo=mean(boot.sqerr)

AARo=mean(boot.abserr)

APEo=mean(boot.perr)

OPsq=.632\*(ASRo-ASR)

OPab=.632\*(AARo-AAR)

OPpe=.632\*(APEo-APE)

RMSEP=sqrt(ASR+OPsq)

MAEP=AAR+OPab

MAPEP=(APE+OPpe)\*100

cat("RMSEP\n")

cat("===============\n")

cat(RMSEP,"\n\n")

cat("MAE\n")

cat("===============\n")

cat(MAEP,"\n\n")

cat("MAPE\n")

cat("===============\n")

cat(MAPEP,"\n\n")

return(data.frame(RMSEP=RMSEP,MAE=MAEP,MAPE=MAPEP))

}

> log.lm1 = lm(Price~.,data=SaratogaTrans) 🡨 model fit using log(Price) as the response   
 and some of the predictors transformed   
 (see examples from Section 2)  
> results = bootlog.cv(log.lm1,B=100)

RMSEP

===============

61337.27

MAE

===============

41569.84

MAPE

===============

0.2430109

> results = bootlog.cv(log.lm1,B=1000)

RMSEP

===============

61061.89

MAE

===============

41508.23

MAPE

===============

0.244127

> results = bootlog.cv(log.lm1,B=5000)

RMSEP

The model using the log transformed response and terms based on predictor transformations outperforms the model fit to these data in the original scale in terms the MAPE (as expected) but also in terms of MAE, though only slightly.

===============

61231.9

MAE

===============

41563.3

MAPE

===============

0.2448597

There have been some recent improvements in the .632 bootstrap procedure, called the .632+ bootstrap which is primarily used for classification problems vs. regression problems. The package sortinghat has functions for calculating the .632+ bootstrap for classification models.

**3.7 – Monte Carlo Cross-validation (MCCV)**

As noted in the previous section the results from the .632 bootstrap will vary from run-to-run, particularly when the number of bootstrap samples () is smaller. This also happens when using split-sample and k-fold cross-validation methods. Using split-sample cross-validation (either training/validation or training/validation/test sets) can lead produce very different results depending which observations in our original data end up in these disjoint sets. The same is true for k-fold cross-validation. The results will vary depending on which observations fall into each of the folds/subsets. In general, the variability from one CV to another, regardless of the method, will increase as the number of observations in the original data set decreases. The example below illustrates this phenomenon for 10-fold CV applied to the Saratoga, NY home prices data.

> results = kfold.MLR(home.lm1)

RMSEP

===============

58765.42

MAEP

===============

41542.15

MAPEP

===============

0.2564174

> results = kfold.MLR(home.lm1)

RMSEP

===============

58993.49

MAEP

===============

41794.73

MAPEP

===============

0.2568648

> results = kfold.MLR(home.lm1)

RMSEP

===============

58657.76

MAEP

===============

41530.58

MAPEP

===============

0.2557584

> results = kfold.MLR(home.lm1)

RMSEP

===============

58697.33

MAEP

===============

41543.52

MAPEP

===============

0.2555822

ETC…

Results from split-sample approaches are even more variable. The function (MLR.ssmc) below will perform repeated , times split-sample cross-validations using training/validation sets. The fraction in the training set is determined by the argument which can certainly be changed.

**Function to perform Split-Sample Monte Carlo (ssmc) cross-validation for MLR**  
  
MLR.ssmc = function(fit,p=.667,M=100,data=fit$model) {

RMSEP = rep(0,M)

MAEP = rep(0,M)

MAPEP = rep(0,M)

y = fit$model[,1]

x = fit$model[,-1]

n = nrow(data)

for (i in 1:M) {

ss = floor(n\*p)

sam = sample(1:n,ss,replace=F)

fit2 = lm(formula(fit),data=data[sam,])

ypred = predict(fit2,newdata=x[-sam,])

RMSEP[i] = sqrt(mean((y[-sam]-ypred)^2))

MAEP[i] = mean(abs(y[-sam]-ypred))

MAPEP[i]=mean(abs(y[-sam]-ypred)/y[-sam])

}

cv = return(data.frame(RMSEP=RMSEP,MAEP=MAEP,MAPEP=MAPEP\*100))

}

To see the variability from one split-sample to the next we can run this function with .

> MLR.ssmc(home.lm1,M=1)

RMSEP MAEP MAPEP

1 57465.56 41476.97 0.2386411

> MLR.ssmc(home.lm1,M=1)

RMSEP MAEP MAPEP

1 64775.52 43779.2 0.3115381

> MLR.ssmc(home.lm1,M=1)

RMSEP MAEP MAPEP

1 58319.74 41420.09 0.3392159

> MLR.ssmc(home.lm1,M=1)

RMSEP MAEP MAPEP

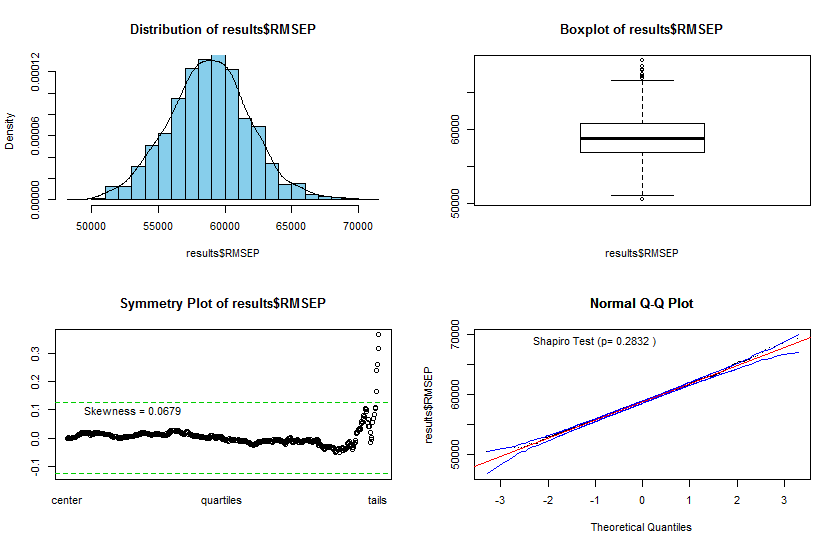
1 58929.07 42660.12 0.2914981

> MLR.ssmc(home.lm1,M=1)

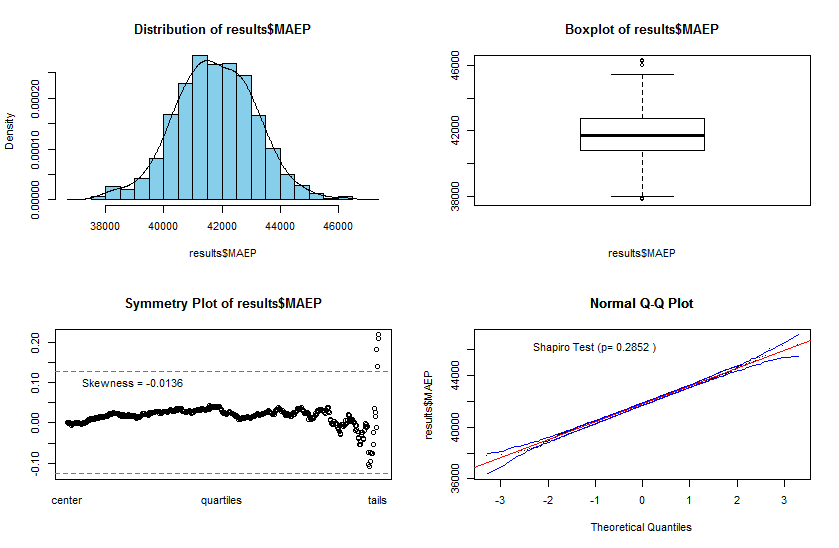
RMSEP MAEP MAPEP

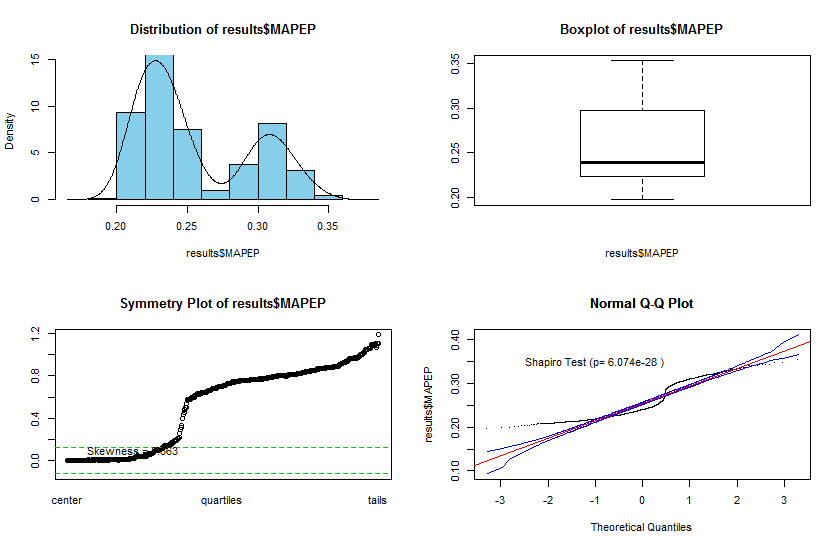
1 60508.61 42360.72 0.2546718

Here we can see the all three measures of predictive performance vary fairly substantially from one split-sample CV to the next. We can better see this by increasing M and examining the distribution of each predictive measure.

> results = MLR.ssmc(home.lm1,M=1000)  
> Statplot(results$RMSEP)  


> Statplot(results$MAEP)



> Statplot(results$MAPEP)  


The bimodal appearance of the mean percent error for prediction (MAPEP) is interesting. It suggests to me that depending which if certain observations end up in the validation set the percent error is greatly increased (over 5-percentage point increase).

Summary statistics for the predictive measures over all split-sample cross-validations are presented below.

> summary(results)

RMSEP MAEP MAPEP

Min. :50498 Min. :37782 Min. :0.1970

1st Qu.:56852 1st Qu.:40828 1st Qu.:0.2238

Median :58804 Median :41746 Median :0.2390

Mean :58817 Mean :41771 Mean :0.2551

3rd Qu.:60733 3rd Qu.:42735 3rd Qu.:0.2970

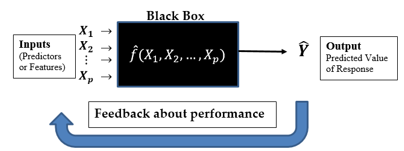
Max. :69219 Max. :46297 Max. :0.3535

To get an accurate measure of predictive performance using k-fold CV, and particularly split-sample approaches, simulation studies have shown that Monte Carlo CV (i.e. random restarts of these CV methods) is recommended as there is inherent variation due to the random allocation of observations to the different subsets. Also note some literature refers to MCCV as the Monte Carlo approach applied to split-sample CV only, although I would argue the MC approach could be applied to any form of cross-validation where random assignment of observations to subsets is used.

**In summary, the methods of Cross-Validation we have discussed are as follows:**

1. **Split-Sample Cross-Validation** - using either training/validation or training/validation/test sets.
2. **k-fold Cross-Validation** - again k = 5 or 10 are typically used.
3. **Leave-Out-One CV** (LOOCV) and **Generalized CV** (GCV)
4. **.632 Bootstrap** (i.e. *resampling methods*)
5. **Monte Carlo Cross-Validation** (MCCV) – which can/should be applied to split-sample and k-fold methods.

Rather than consider more examples at this point we will move on to more methods for estimating . Cross-validation will be continually used throughout the course to fine-tune our models and choose between competing modeling approaches.



**Cross-Validation methods are used for this purpose!**

**More on Prediction Error and the Variance-Bias Tradeoff and the Role of CV**

For any regression problem we assume that the response has the following model:

where = and .

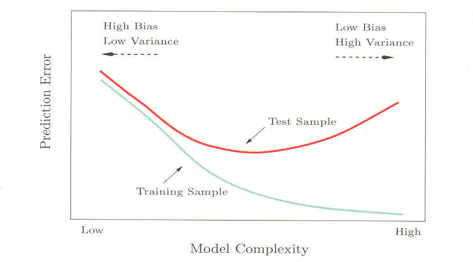
Our goal in modeling is to approximate or estimate using a random sample of size *n* from the population*:*  where the are the observed values of the *p*-dimensional predictor vectors and the are the corresponding observed values of the response.

= +

=

The cross-validation methods discussed above are all acceptable ways to estimate , but some are certainly better than others. This is still an active area of research and there is no definitive best method for every situation. Some methods are better at estimating the variance component of the while others are better at estimating the bias. Ideally we would like to use a method of cross-validation that does a reasonable job of estimating each component.

In the sections that follow we will be introducing alternatives to OLS or variations of OLS for developing models to estimate . Some of these modeling strategies have the potential to be very flexible (i.e. have small bias) but potentially at the expense of being highly variable, i.e. have large variation, . Balancing these two components of squared prediction error is critical and cross-validation is one of the main tools we will use to create this balance in our model development process.



Validation or Test Sample

The figure is taken from pg. 194 of *The Elements of Statistical Learning* by Hastie, Tibshirani, and Friedman.