**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 there are several methods for estimating a model (e.g. MLR, 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.

**Example 3.1 – Diamond 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.

> Diamonds = read.table(file.choose(),header=T,sep=”,”)



> names(Diamonds)

 [1] "Price" "Carats" "Color" "Clarity" "Depth" "Table"

 [7] "Cut" "TDdiff" "TDratio" "Test"

> Diamonds$Price = log(Diamonds$Price)

> Diamonds = Diamonds[,-c(5,6,10)]

> str(Diamonds)

'data.frame': 2690 obs. of 7 variables:

 $ Price : num 6.91 6.91 6.91 6.91 6.91 ...

 $ Carats : num 0.3 0.44 0.31 0.66 0.47 0.4 0.36 0.52 0.53 0.43 ...

 $ Color : Factor w/ 8 levels "D","E","F","G",..: 2 2 2 8 5 4 1 5 1 3 ...

 $ Clarity: Factor w/ 7 levels "IF","SI1","SI2",..: 6 5 6 2 5 4 5 3 3 5 ...

 $ Cut : Factor w/ 4 levels "Excellent","Good",..: 1 1 1 1 4 1 1 4 4 1 ...

 $ TDdiff : num -1 -3.9 -3.3 -5.8 4.9 -3 -4.3 -0.7 -0.4 -1.5 ...

 $ TDratio: num 0.983 0.937 0.946 0.908 1.083 ...

> dim(Diamonds) 🡨 There are n = 2690 diamonds for use in developing our model

[1] 2690 7

> n = nrow(Diamonds)

> n

[1] 2690

**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,floor(.66666\*n),replace=F)

> length(train)

[1] 1793

> .66666\*n

[1] 1793.315

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

> Diamonds.train = Diamonds[train,]

> Diamonds.valid = Diamonds[-train,]

> dim(Diamonds.train)

[1] 1793 7

> dim(Diamonds.valid)

[1] 897 7

**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(Diamonds)

> m1 = floor(n\*.60)

> m2 = floor(n\*.20)

> RO = sample(1:n,size=n,replace=F) 🡨 this commands 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] 1614

> length(valid)

[1] 539

> length(test)

[1] 537

> 1614+539+537

[1] 2690

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,]

Diamonds.train = Diamonds[train,]

Diamonds.valid = Diamonds[valid,]

Diamonds.test = Diamonds[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. You used this function in evaluating your MLR model for diamond prices.

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 Diamonds data.

We will first fit a model using log(Price) as the response and all of the potential predictors in their original scale.

> diam.lm1 = lm(Price~.,data=Diamonds.train)

> summary(diam.lm1)

Call:

lm(formula = Price ~ Carats + Clarity + Cut + Color + TDdiff +

 TDratio, data = Diamonds.train)

Residuals:

 Min 1Q Median 3Q Max

-0.94350 -0.13028 0.01388 0.12664 0.63206

Coefficients:

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

(Intercept) 5.78699 4.76855 1.214 0.22509

Carats 2.33298 0.01991 117.166 < 2e-16 \*\*\*

ClaritySI1 -0.38304 0.02540 -15.082 < 2e-16 \*\*\*

ClaritySI2 -0.53822 0.02623 -20.516 < 2e-16 \*\*\*

ClarityVS1 -0.19315 0.02588 -7.463 1.39e-13 \*\*\*

ClarityVS2 -0.25718 0.02560 -10.045 < 2e-16 \*\*\*

ClarityVVS1 -0.11511 0.02646 -4.351 1.44e-05 \*\*\*

ClarityVVS2 -0.06646 0.02683 -2.477 0.01337 \*

CutGood -0.07173 0.02279 -3.147 0.00168 \*\*

CutIdeal 0.04273 0.02103 2.032 0.04235 \*

CutVery Good -0.02906 0.01162 -2.500 0.01253 \*

ColorE -0.04150 0.01907 -2.176 0.02970 \*

ColorF -0.08001 0.01997 -4.006 6.45e-05 \*\*\*

ColorG -0.13505 0.02054 -6.573 6.65e-11 \*\*\*

ColorH -0.21910 0.02054 -10.668 < 2e-16 \*\*\*

ColorI -0.36887 0.02221 -16.604 < 2e-16 \*\*\*

ColorJ -0.54175 0.02376 -22.802 < 2e-16 \*\*\*

ColorK -0.77575 0.02922 -26.552 < 2e-16 \*\*\*

TDdiff -0.01192 0.07650 -0.156 0.87615

TDratio 0.76185 4.77089 0.160 0.87315

---

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

Residual standard error: 0.1989 on 1594 degrees of freedom

Multiple R-squared: 0.9109, Adjusted R-squared: 0.9099

F-statistic: 858.1 on 19 and 1594 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.

> yact = exp(Diamonds.valid$Price)

> ypred = exp(predict(diam.lm1,newdata=Diamonds.valid))

> results = PredAcc(yact,ypred)

RMSEP

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

1240.649

MAE

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

702.1419

MAPE

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

15.50811

> results

 RMSEP MAE MAPE

1 1240.649 702.1419 15.50811

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] 1240.649

> results$MAE

[1] 702.1419

> results$MAPE

[1] 15.50811

We will now construct a model that incorporates a cubic polynomial in carats.

> diam.lm2 = lm(Price~poly(Carats,3)+Clarity+Color+Cut+TDdiff+TDratio,data=Diamonds.train)

> summary(diam.lm2)

Call:

lm(formula = Price ~ poly(Carats, 3) + Clarity + Color + Cut +

 TDdiff + TDratio, data = Diamonds.train)

Residuals:

 Min 1Q Median 3Q Max

-0.43744 -0.07604 -0.00225 0.07010 0.38732

Coefficients:

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

(Intercept) 12.123380 2.718716 4.459 8.80e-06 \*\*\*

poly(Carats, 3)1 32.009015 0.151538 211.227 < 2e-16 \*\*\*

poly(Carats, 3)2 -6.979578 0.121165 -57.604 < 2e-16 \*\*\*

poly(Carats, 3)3 0.534976 0.117295 4.561 5.48e-06 \*\*\*

ClaritySI1 -0.560535 0.014905 -37.607 < 2e-16 \*\*\*

ClaritySI2 -0.698530 0.015345 -45.521 < 2e-16 \*\*\*

ClarityVS1 -0.306861 0.014925 -20.560 < 2e-16 \*\*\*

ClarityVS2 -0.422515 0.014943 -28.276 < 2e-16 \*\*\*

ClarityVVS1 -0.109163 0.015072 -7.243 6.82e-13 \*\*\*

ClarityVVS2 -0.183431 0.015511 -11.826 < 2e-16 \*\*\*

ColorE -0.068411 0.010875 -6.291 4.07e-10 \*\*\*

ColorF -0.123588 0.011401 -10.840 < 2e-16 \*\*\*

ColorG -0.202177 0.011763 -17.188 < 2e-16 \*\*\*

ColorH -0.300328 0.011785 -25.484 < 2e-16 \*\*\*

ColorI -0.425448 0.012693 -33.518 < 2e-16 \*\*\*

ColorJ -0.592037 0.013563 -43.650 < 2e-16 \*\*\*

ColorK -0.781806 0.016724 -46.746 < 2e-16 \*\*\*

CutGood -0.103315 0.013017 -7.937 3.88e-15 \*\*\*

CutIdeal 0.018498 0.011991 1.543 0.123

CutVery Good -0.034293 0.006629 -5.173 2.60e-07 \*\*\*

TDdiff 0.056343 0.043605 1.292 0.196

TDratio -3.360776 2.719381 -1.236 0.217

---

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

Residual standard error: 0.1133 on 1592 degrees of freedom

Multiple R-squared: 0.9711, Adjusted R-squared: 0.9708

F-statistic: 2550 on 21 and 1592 DF, p-value: < 2.2e-16

> ypred = exp(predict(diam.lm2,newdata=Diamonds.valid))

> results2 = PredAcc(yact,ypred)

RMSEP

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

521.9644

MAE

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

339.686

MAPE

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

8.199608

Next we consider a model containing cubic polynomial terms in carat size, clarity, color, cut, TDdiff, TDratio, and pairwise interaction terms between the ordinal variables (clarity, color, and cut).

> diam.lm3 = lm(Price~poly(Carats,3)+Clarity\*Color+Clarity\*Cut+Color\*Cut+TDdiff+TDratio,
data=Diamonds.train)

> summary(diam.lm3)

Call:

lm(formula = Price ~ poly(Carats, 3) + Clarity \* Color + Clarity \*

 Cut + Color \* Cut + TDdiff + TDratio, data = Diamonds.train)

Residuals:

 Min 1Q Median 3Q Max

-0.40292 -0.06748 -0.00108 0.06528 0.34137

Coefficients:

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

(Intercept) 9.007e+00 2.515e+00 3.582 0.000352 \*\*\*

poly(Carats, 3)1 3.238e+01 1.421e-01 227.831 < 2e-16 \*\*\*

poly(Carats, 3)2 -7.592e+00 1.168e-01 -64.978 < 2e-16 \*\*\*

poly(Carats, 3)3 4.904e-01 1.118e-01 4.385 1.24e-05 \*\*\*

ClaritySI1 -8.270e-01 4.061e-02 -20.364 < 2e-16 \*\*\*

ClaritySI2 -9.505e-01 4.174e-02 -22.774 < 2e-16 \*\*\*

ClarityVS1 -4.858e-01 3.951e-02 -12.297 < 2e-16 \*\*\*

ClarityVS2 -6.167e-01 3.837e-02 -16.071 < 2e-16 \*\*\*

ClarityVVS1 -2.435e-01 4.004e-02 -6.081 1.51e-09 \*\*\*

ClarityVVS2 -3.088e-01 3.771e-02 -8.190 5.51e-16 \*\*\*

ColorE -2.577e-01 3.999e-02 -6.443 1.57e-10 \*\*\*

... ... ... ... ...

ColorE:CutVery Good -1.086e-02 2.160e-02 -0.503 0.615158

ColorF:CutVery Good -3.118e-02 2.264e-02 -1.377 0.168705

ColorG:CutVery Good -8.134e-03 2.301e-02 -0.353 0.723798

ColorH:CutVery Good -2.261e-02 2.304e-02 -0.981 0.326533

ColorI:CutVery Good 3.348e-03 2.427e-02 0.138 0.890274

ColorJ:CutVery Good 3.567e-04 2.523e-02 0.014 0.988723

ColorK:CutVery Good -1.626e-05 3.053e-02 -0.001 0.999575

---

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

Residual standard error: 0.1007 on 1511 degrees of freedom

Multiple R-squared: 0.9783, Adjusted R-squared: 0.9769

F-statistic: 669.3 on 102 and 1511 DF, p-value: < 2.2e-16

> ypred = exp(predict(diam.lm3,newdata=Diamonds.valid))

> results3 = PredAcc(yact,ypred)

RMSEP

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

473.9746

MAE

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

326.6201

MAPE

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

8.117986

Even though this model (diam.lm3) is more complex, it does have slightly better predictive performance than the model without the interaction terms between the ordinal predictors.

Next we use backwards elimination to eliminate some of the nonsignificant terms from this model.

> diam.step = step(diam.lm3)

Start: AIC=-7309.99

Price ~ poly(Carats, 3) + Clarity \* Color + Clarity \* Cut + Color \* Cut + TDdiff + TDratio

 Df Sum of Sq RSS AIC

- Color:Cut 21 0.24 15.56 -7327.4

- Clarity:Cut 18 0.24 15.56 -7321.3

- TDratio 1 0.00 15.33 -7312.0

- TDdiff 1 0.00 15.33 -7312.0

<none> 15.33 -7310.0

- Clarity:Color 42 4.52 19.85 -6976.9

- poly(Carats, 3) 3 538.84 554.17 -1525.4

Step: AIC=-7327.42

Price ~ poly(Carats, 3) + Clarity + Color + Cut + TDdiff + TDratio +

 Clarity:Color + Clarity:Cut

 Df Sum of Sq RSS AIC

- Clarity:Cut 18 0.25 15.82 -7337.3

- TDratio 1 0.00 15.56 -7329.4

- TDdiff 1 0.00 15.56 -7329.3

<none> 15.56 -7327.4

- Clarity:Color 42 4.64 20.20 -6990.6

- poly(Carats, 3) 3 547.25 562.81 -1542.4

Step: AIC=-7337.31

Price ~ poly(Carats, 3) + Clarity + Color + Cut + TDdiff + TDratio +

 Clarity:Color

 Df Sum of Sq RSS AIC

- TDratio 1 0.00 15.82 -7339.1

- TDdiff 1 0.00 15.82 -7339.0

<none> 15.82 -7337.3

- Cut 3 0.82 16.64 -7261.6

- Clarity:Color 42 4.62 20.43 -7008.0

- poly(Carats, 3) 3 557.21 573.03 -1549.4

Step: AIC=-7339.09

Price ~ poly(Carats, 3) + Clarity + Color + Cut + TDdiff + Clarity:Color

 Df Sum of Sq RSS AIC

<none> 15.82 -7339.1

- TDdiff 1 0.06 15.88 -7334.5

- Cut 3 1.04 16.86 -7241.9

- Clarity:Color 42 4.63 20.45 -7008.4

- poly(Carats, 3) 3 558.67 574.49 -1547.2

> summary(diam.step)

Residual standard error: 0.101 on 1551 degrees of freedom

Multiple R-squared: 0.9777, Adjusted R-squared: 0.9768

F-statistic: 1094 on 62 and 1551 DF, p-value: < 2.2e-16

> ypred = exp(predict(diam.step,newdata=Diamonds.valid))

> results.step = PredAcc(yact,ypred)

RMSEP

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

475.3449

MAE

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

323.6816

MAPE

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

7.998595

The simpler model has better predictive performance than the larger, more complex (40 more parameters), MLR model. At this point we might decide this reduced model is the “best” MLR model we can develop for these data (which I 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 = exp(predict(diam.step,newdata=Diamonds.test))

> yact = exp(Diamonds.test$Price)

> results.test = PredAcc(yact,ypred)

RMSEP

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

511.4938

MAE

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

341.212

MAPE

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

8.176525

These results could now be reported as the expected predictive accuracy of our MLR model as we move forward predicting the selling price of future diamonds given these data characteristics (i.e. predictor values).

It is important to note that though the model was only fit using the training cases and we compared rival models using the prediction accuracy for the validation cases. Thus only the training and 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 when the response has been log transformed.**

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

 sum.sqerr = rep(0,k)

 sum.abserr = rep(0,k)

 sum.pererr = rep(0,k)

 n = nrow(data)

 y = exp(fit$model[,1])

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

 for (i in 1:k) {

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

 ypred = exp(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=100\*sum(sum.pererr)/n))

}

Comments on the kfold.MLR code:

Below the kfold.MLR function is used to compare the different models we fit above for predicting diamond prices. As we will be cross-validating via k-fold CV we will use the full dataset when fitting these models.

> diam.lm1 = lm(formula(diam.lm1),data=Diamonds)
> diam.lm2 = lm(formula(diam.lm2),data=Diamonds)

> diam.lm3 = lm(formula(diam.lm3),data=Diamonds)

> diam.step = lm(formula(diam.step),data=Diamonds)

> kfold.MLR(diam.lm1,data=Diamonds)

 RMSEP MAE MAPE

1 1354.914 745.6321 16.08141

> kfold.MLR(diam.lm2,data=Diamonds)

 RMSEP MAE MAPE

1 569.1011 363.8074 8.740581

> kfold.MLR(diam.lm3,data=Diamonds)

 RMSEP MAE MAPE

1 500.2554 330.4602 8.198914

> kfold.MLR(diam.step,data=Diamonds)

 RMSEP MAE MAPE

1 491.8918 324.9147 8.08469

Again we see that the stepwise reduced version of the third model has the best predictive performance.

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

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 our “best” model from diamonds

> kfold.MLR(diam.step,data=Diamonds)

 RMSEP MAE MAPE

1 494.4699 325.1654 8.083181

> kfold.MLR(diam.step,data=Diamonds)

 RMSEP MAE MAPE

1 493.2146 325.4464 8.093104

> kfold.MLR(diam.step,data=Diamonds)

 RMSEP MAE MAPE

1 496.4355 326.5009 8.102121

> kfold.MLR(diam.step,data=Diamonds)

 RMSEP MAE MAPE

1 495.4469 326.4109 8.10649

> kfold.MLR(diam.step,data=Diamonds)

 RMSEP MAE MAPE

1 497.2608 327.0385 8.117589

ETC…

Results from split-sample approaches can be even more variable. The function (MLR.sscv) 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.sscv = function(fit,data,p=.667,M=100) {

 RMSEP = rep(0,M)

 MAEP = rep(0,M)

 MAPEP = rep(0,M)

 y = exp(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 = exp(predict(fit2,newdata=data[-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])\*100

 }

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

}

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

> MLR.sscv(diam.step,data=Diamonds,M=1)

 RMSEP MAEP MAPEP

1 522.4084 344.2979 8.195881

> MLR.sscv(diam.step,data=Diamonds,M=1)

 RMSEP MAEP MAPEP

1 486.8591 326.8938 8.126649

> MLR.sscv(diam.step,data=Diamonds,M=1)

 RMSEP MAEP MAPEP

1 505.4992 327.1596 8.137339

> MLR.sscv(diam.step,data=Diamonds,M=1)

 RMSEP MAEP MAPEP

1 496.2471 326.0286 8.423541

> MLR.sscv(diam.step,data=Diamonds,M=1)

 RMSEP MAEP MAPEP

1 473.3669 313.3284 7.929003

> MLR.sscv(diam.step,data=Diamonds,M=1)

 RMSEP MAEP MAPEP

1 501.4451 340.6297 8.479095

> MLR.sscv(diam.step,data=Diamonds,M=1)

 RMSEP MAEP MAPEP

1 493.0344 326.8548 7.986915

> MLR.sscv(diam.step,data=Diamonds,M=1)

 RMSEP MAEP MAPEP

1 484.6854 321.0687 8.120754

Here we can see the all three measures of predictive performance vary 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.sscv(diam.step,Diamonds,M=1000)
> hist(results$RMSEP)

> hist(results$MAE)

> hist(results$MAPE)



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

> summary(results)

 RMSEP MAEP MAPEP

 Min. :434.8 Min. :297.2 Min. :7.555

 1st Qu.:482.4 1st Qu.:320.4 1st Qu.:8.017

 Median :497.2 Median :328.1 Median :8.148

 Mean :498.3 Mean :328.1 Mean :8.143

 3rd Qu.:512.7 3rd Qu.:335.5 3rd Qu.:8.270

 Max. :581.2 Max. :366.4 Max. :8.753

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 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. **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 used throughout 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.