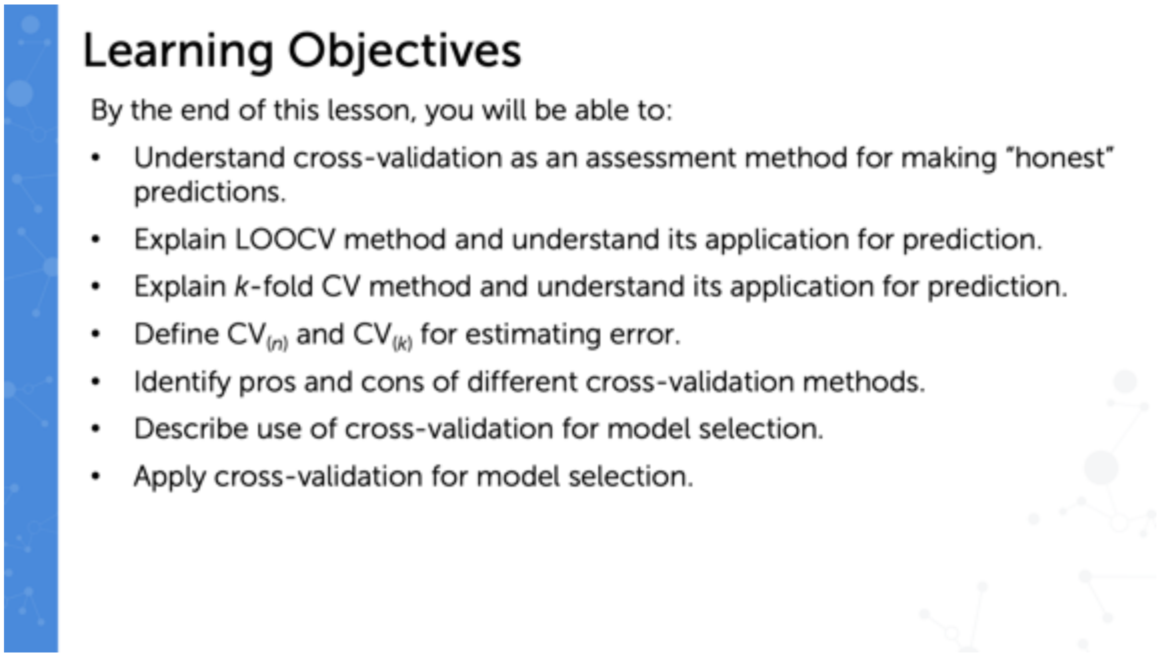
UNIVERSITY OF WISCONSIN
DATA SCIENCE

DS 740

Data Mining

Cross-Validation for Honest Prediction & Model Selection
Using Validation Sets Cleverly

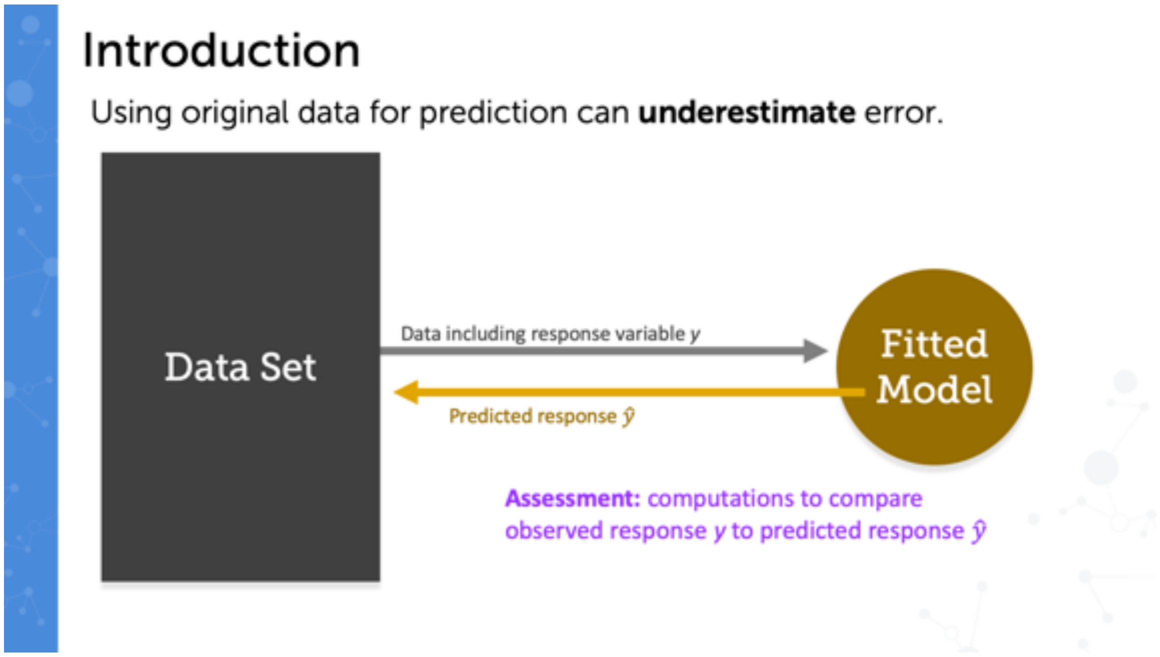
Important note: Transcripts are **not** substitutes for textbook assignments.



Learning Objectives

By the end of this lesson, you will be able to:

- Understand cross-validation as an assessment method for making "honest" predictions.
- Explain LOOCV method and understand its application for prediction.
- Explain *k*-fold CV method and understand its application for prediction.
- Define $CV_{(n)}$ and $CV_{(k)}$ for estimating error.
- Identify pros and cons of different cross-validation methods.
- Describe use of cross-validation for model selection.
- Apply cross-validation for model selection.



Introduction

Using original data for prediction can **underestimate** error.

Data Set
Data including response variable y
Fitted Model
Predicted response $\hat{y}$
Assessment: computations to compare observed response y to predicted response $\hat{y}$

We assess a model by predicting data and summarizing the error in prediction. Estimating the error of a model on the same data that were used to build or train the model can give an underestimate of the error. We are not able to get an honest picture of how well the model can actually predict new data when we reuse the data used to fit the model.

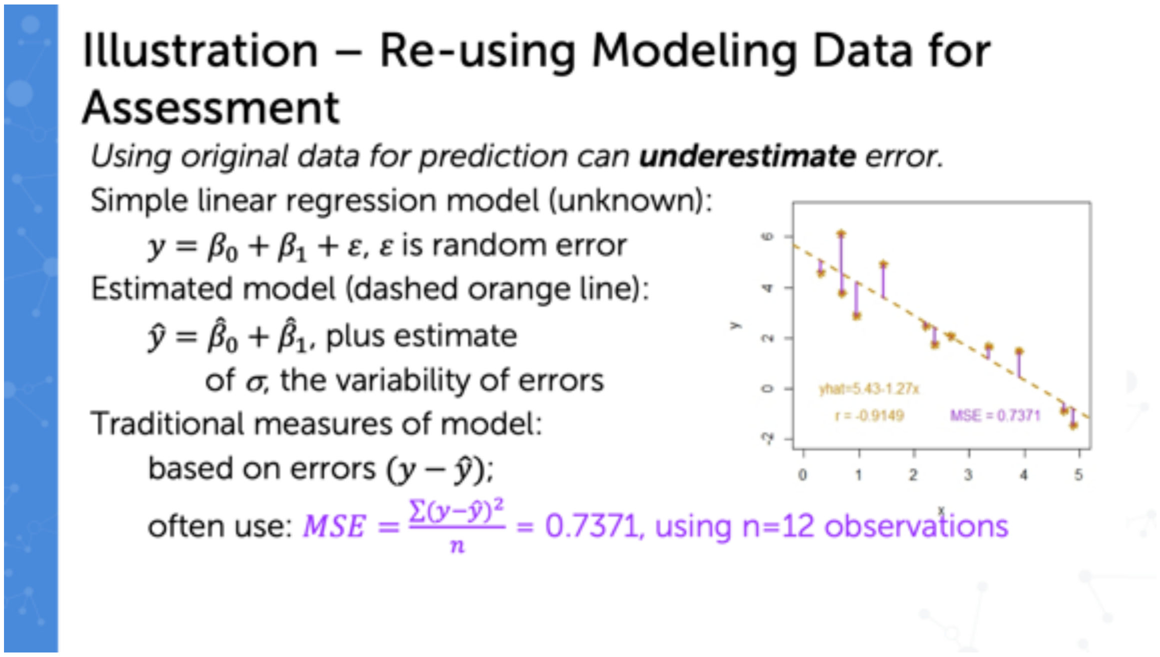


Illustration – Re-using Modeling Data for Assessment

*Using original data for prediction can **underestimate** error.*

Simple linear regression model (unknown):

$y = \beta_0 + \beta_1 + \varepsilon$, ε is random error

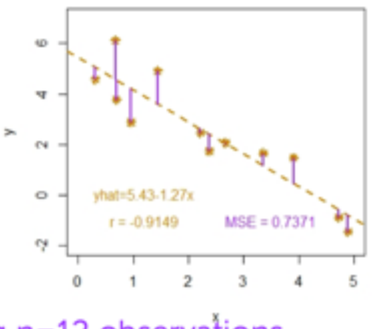
Estimated model (dashed orange line):

$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1$, plus estimate of σ, the variability of errors

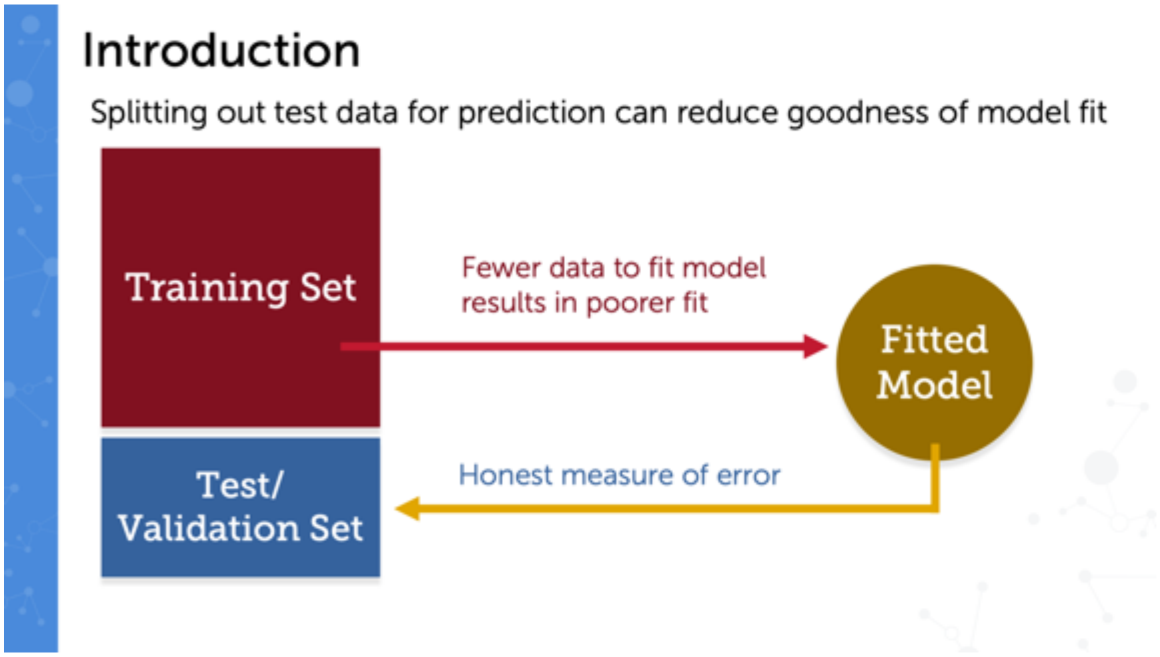
Traditional measures of model:

based on errors $(y - \hat{y})$;

often use: $MSE = \frac{\Sigma(y-\hat{y})^2}{n}$ = 0.7371, using n=12 observations



Let's take a look at an example. Consider a simple linear regression model in which we are estimating the intercept and slope of the best fitting line to xy data. Our purpose is to produce a line which can be used to predict the response based on the observed predictor, x. Thus, we choose the line to minimize the distance that the points fall from the line, or more specifically, a summary of the distance that the points fall from the line. We work to minimize the SSE, which stands for Sum of Squared Errors, or equivalently, the Mean Square Error, MSE.



Introduction

Splitting out test data for prediction can reduce goodness of model fit

You've seen that the validation set approach lets us estimate what the error of a model will be on new data points in a test or validation set. However, the error varies based on which points get included in that validation set, and this can be an overestimate of the error because the model is being trained on a smaller data set. That is, we are producing a worse model fitted to the data because it's based on fewer data. In this lesson, we'll learn how to avoid those problems.

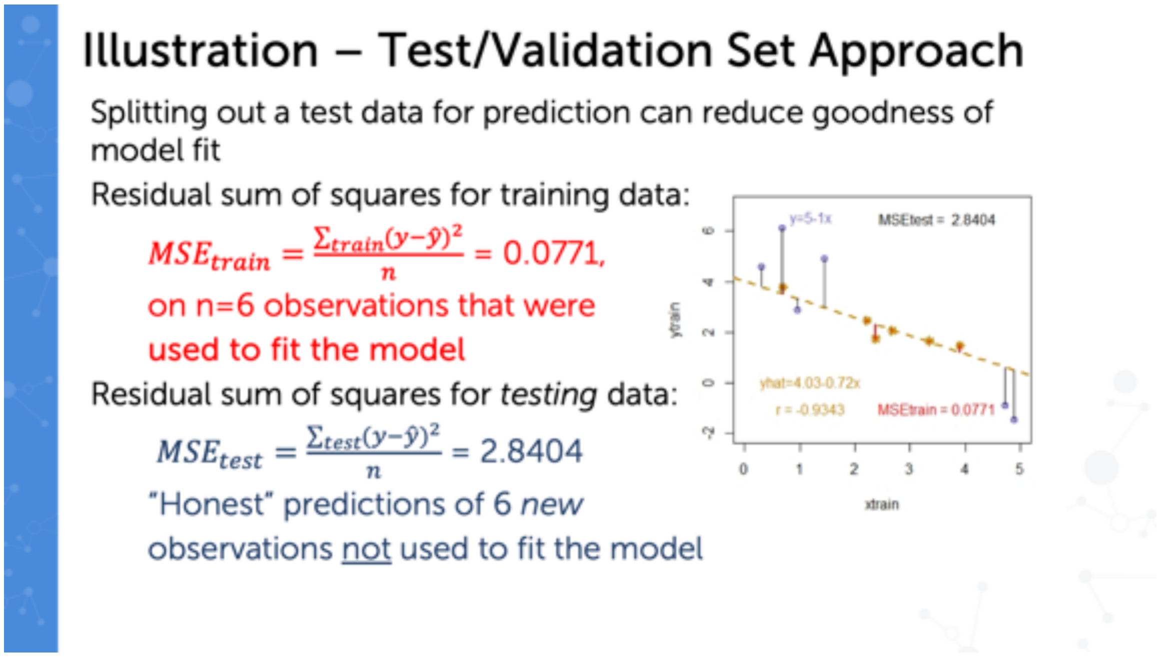


Illustration – Test/Validation Set Approach

Splitting out a test data for prediction can reduce goodness of model fit

Residual sum of squares for training data:

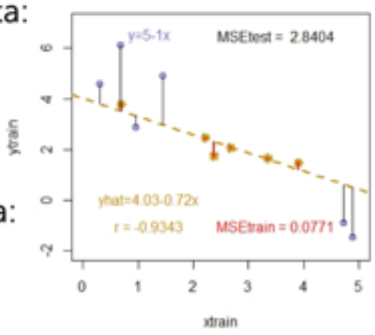
$$MSE_{train} = \frac{\sum_{train}(y-\hat{y})^2}{n} = 0.0771,$$

on n=6 observations that were used to fit the model

Residual sum of squares for *testing* data:

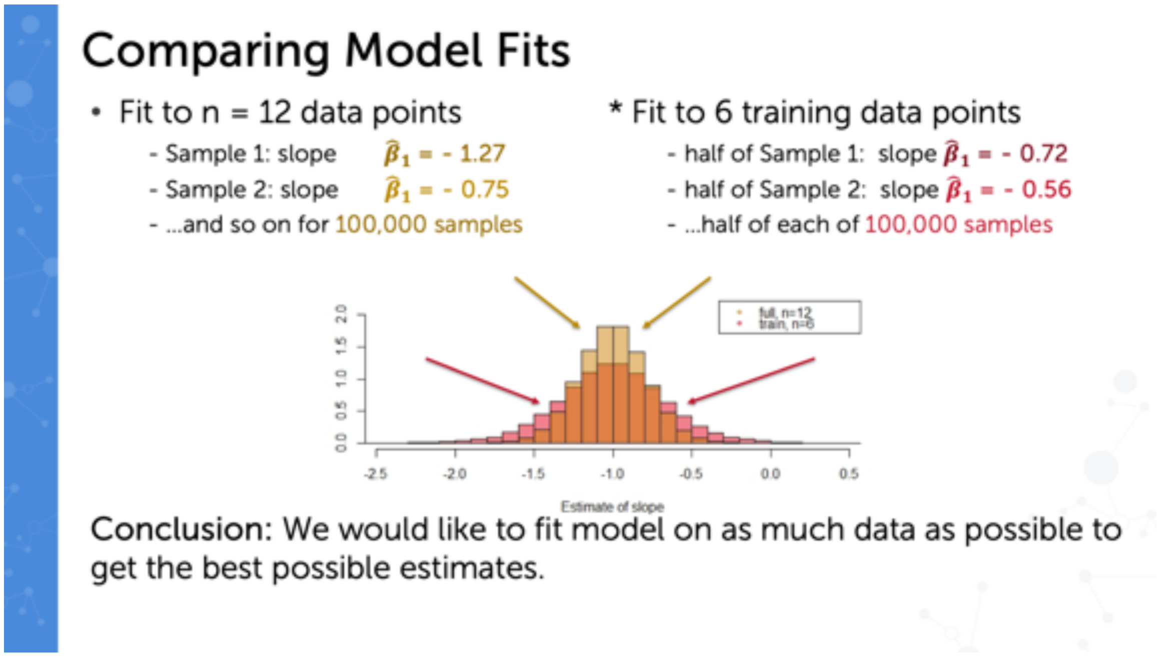
$$MSE_{test} = \frac{\sum_{test}(y-\hat{y})^2}{n} = 2.8404$$

"Honest" predictions of 6 *new* observations not used to fit the model



We illustrate this point by applying a validation set approach on the previous data. We split the data in half with n equals 6, training observations, and 6, testing observations. In this particular example, we are able to fit a line very closely to the training set. However, the prediction of the test set is much worse, and we can see this from the much higher MSE computed on the testing set.

Why is this? Occurs because the test set data were not used to fit the model. And this is the real situation that we wish to mimic when we assess how well a model can do future prediction of new data. That is, how well can it predict data that were not used to fit the model.



Comparing Model Fits

- Fit to n = 12 data points
  - Sample 1: slope $\hat{\beta}_1$ = - 1.27
  - Sample 2: slope $\hat{\beta}_1$ = - 0.75
  - ...and so on for 100,000 samples

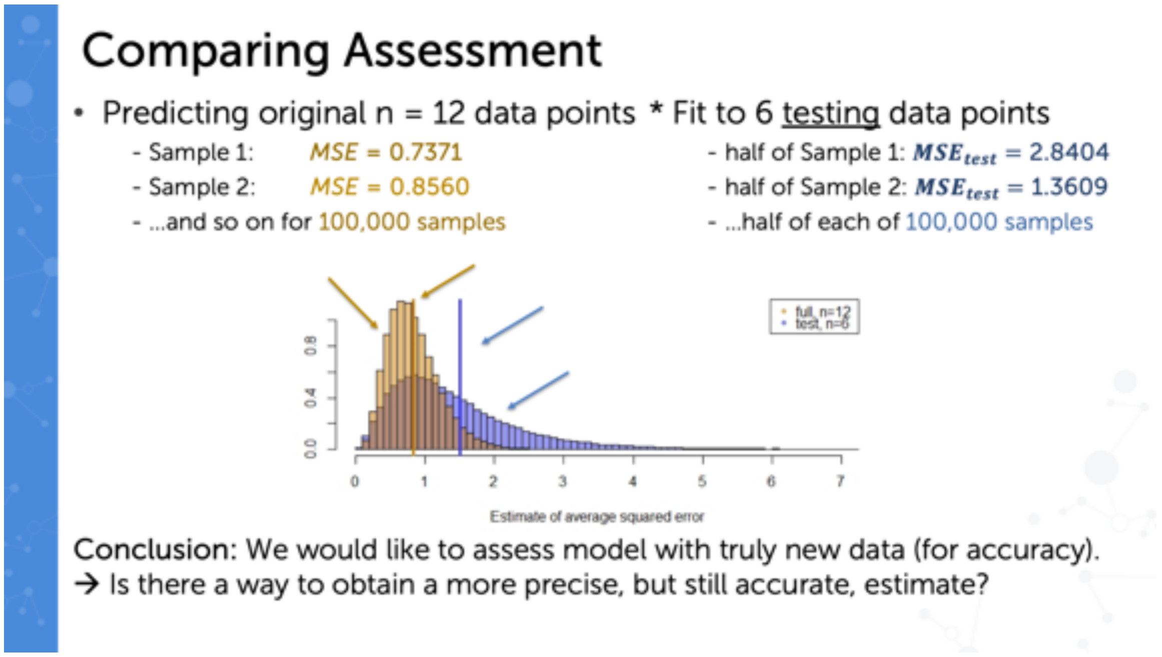
* Fit to 6 training data points
  - half of Sample 1: slope $\hat{\beta}_1$ = - 0.72
  - half of Sample 2: slope $\hat{\beta}_1$ = - 0.56
  - ...half of each of 100,000 samples

Conclusion: We would like to fit model on as much data as possible to get the best possible estimates.

The example in the previous slide was rather extreme, but it suggests some basic principles about model fitting and model assessment that we wish to highlight. One way to illustrate this is via simulation. We consider repeating the process on the previous slides many, many times.

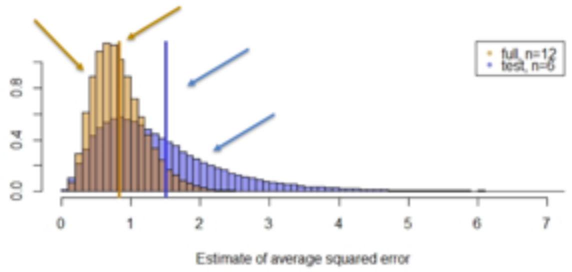
So each time, we sample a total of n equals 12 data points, we fit a simple linear regression model to the full 12 data points, and we then split the data into a training set of size 6, and a testing set of size 6 on which we fit a simple linear regression model to the 6 training observations and predict the 6 testing observations. We repeat this whole process many, many times, say, 100,000, and in doing so, we get 100,000 simple linear regression models fit to n equals 12 observations with 100,000 slopes, that is, a different slope for each of the different sample fits. This is the histogram displayed in a lighter orangish color with a taller peak. We also get 100,000 fitted models to the smaller training set, which again, produces 100,000 slopes, one for each of the fits to each training set. And this is visualized in the reddish wider histogram displaying those 100,000 estimated slopes.

The purpose of this illustration is to show that the smaller sized training data set tends to be less accurate for estimating the slope than is the full data set with n equals 12. While this is intuitively apparent, this visually affirms that intuition. That is, more data produces estimates that are closer, that is, less spread out, to the true value of the slope. Hence, we would like to fit the model on as much data as possible to get our best estimates.



Comparing Assessment

- Predicting original n = 12 data points * Fit to 6 testing data points
  - Sample 1: *MSE = 0.7371* - half of Sample 1: $\boldsymbol{MSE_{test}} = 2.8404$
  - Sample 2: *MSE = 0.8560* - half of Sample 2: $\boldsymbol{MSE_{test}} = 1.3609$
  - ...and so on for 100,000 samples - ...half of each of 100,000 samples

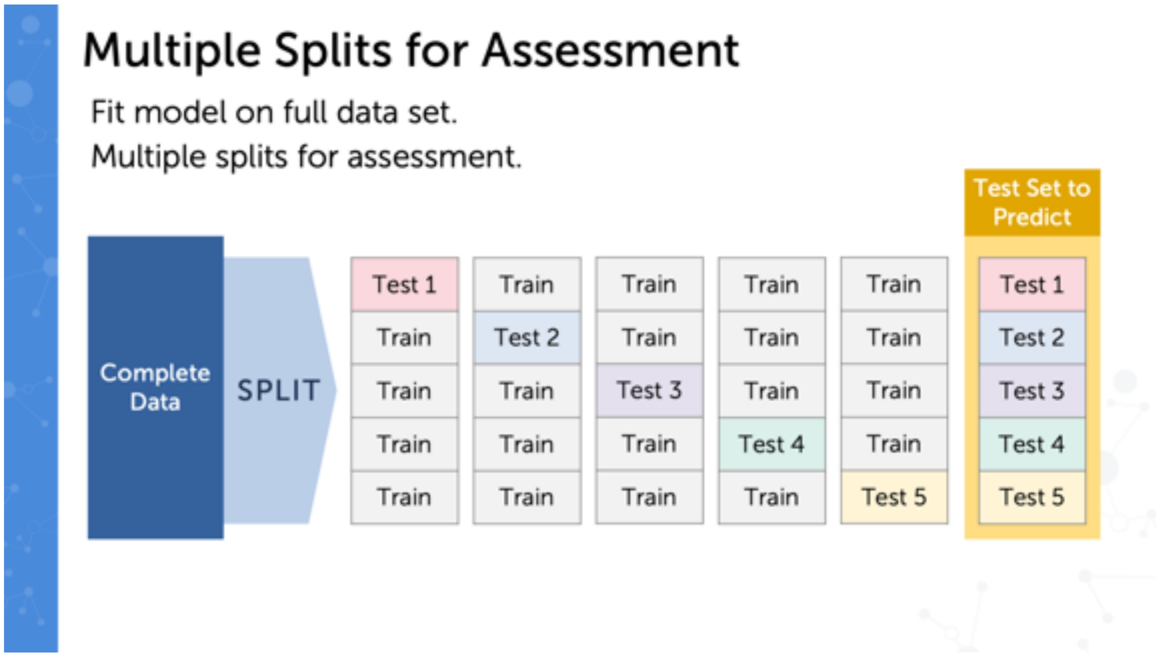


Conclusion: We would like to assess model with truly new data (for accuracy).
→ Is there a way to obtain a more precise, but still accurate, estimate?

We revisit the simulations that were described in the previous slide, looking at a more relevant summary to the topic of discussion, and that is the mean square error. We first take a look at the mean square error values where each is computed on a sample of size n equals 12 by repredicting that full data set using the model fit to the same data. These mean square error values over 100,000 such samples are shown in the yellow lighter histogram, which has a center around 0.7.

We also make a histogram of mean square values computed on the test sets where the model was fit to a training set of 6 values. This is shown in the histogram in the bluish color, which is centered around 1.5. The purpose of comparing these histograms is to show that reusing data, that is, repredicting the same data used to fit the model, will underestimate error, as shown in the lower center for the yellow histogram.

The honest prediction is the one we want, and that is illustrated as being centered right around 1.5. And this shows a more reliable reflection of the true error and estimation. Hence, we would like to assess the model with truly new data for accuracy. However, we also note that the mean square error values from the testing sets tend to be more variable than if we could use more data. So is there a way to obtain a more precise, but still accurate estimate?



Multiple Splits for Assessment

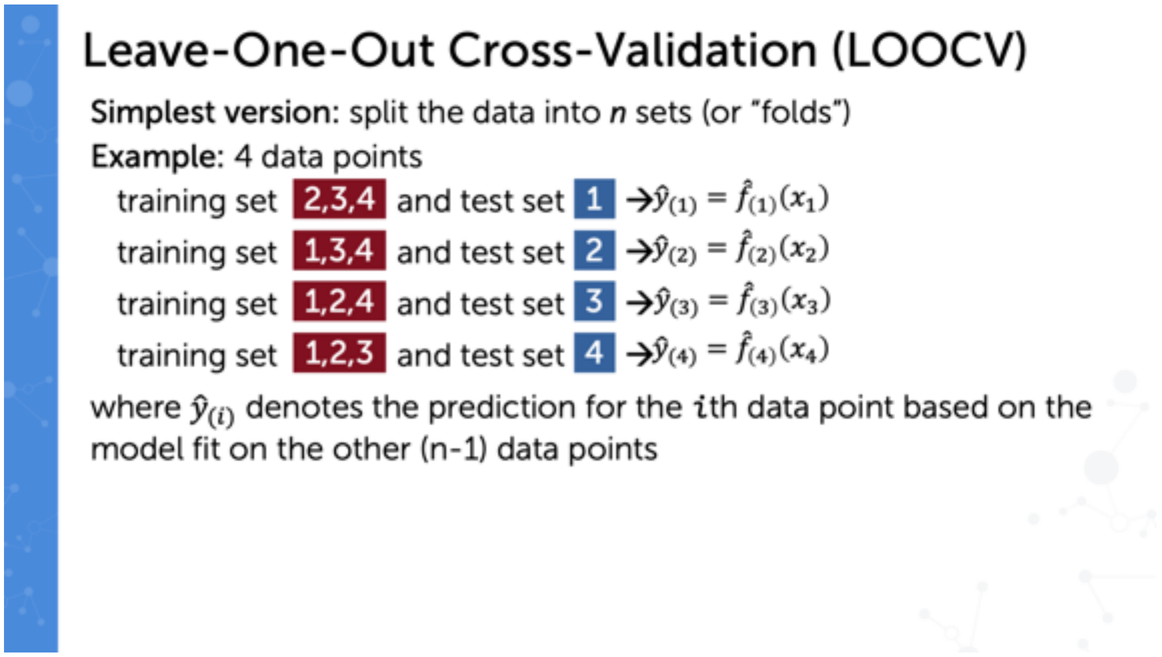
Fit model on full data set.
Multiple splits for assessment.

Test 1	Train	Train	Train	Train	Test 1
Train	Test 2	Train	Train	Train	Test 2
Train	Train	Test 3	Train	Train	Test 3
Train	Train	Train	Test 4	Train	Test 4
Train	Train	Train	Train	Test 5	Test 5

Suppose we were to use all the data to fit the model. We now are left with no data to assess unless we are clever about reusing the data. If we are willing to do some additional model fitting on subsets, we can use these data for assessment as well. Such an assessment process could be accomplished as follows.

Split the data sets into approximately equally sized subsets. In turn, hold each subset out and fit the model on the remaining subsets. Predict the observations in the holdout set.

We are then using a model fit on training set data to predict testing set data. That is, we predict data not used to fit the model. Of course, this does not come for free. We must fit the model additional times, the same number of times as we split the data.


Leave-One-Out Cross-Validation (LOOCV)

Simplest version: split the data into *n* sets (or "folds")

Example: 4 data points

training set 2,3,4 and test set 1 $\rightarrow \hat{y}_{(1)} = \hat{f}_{(1)}(x_1)$

training set 1,3,4 and test set 2 $\rightarrow \hat{y}_{(2)} = \hat{f}_{(2)}(x_2)$

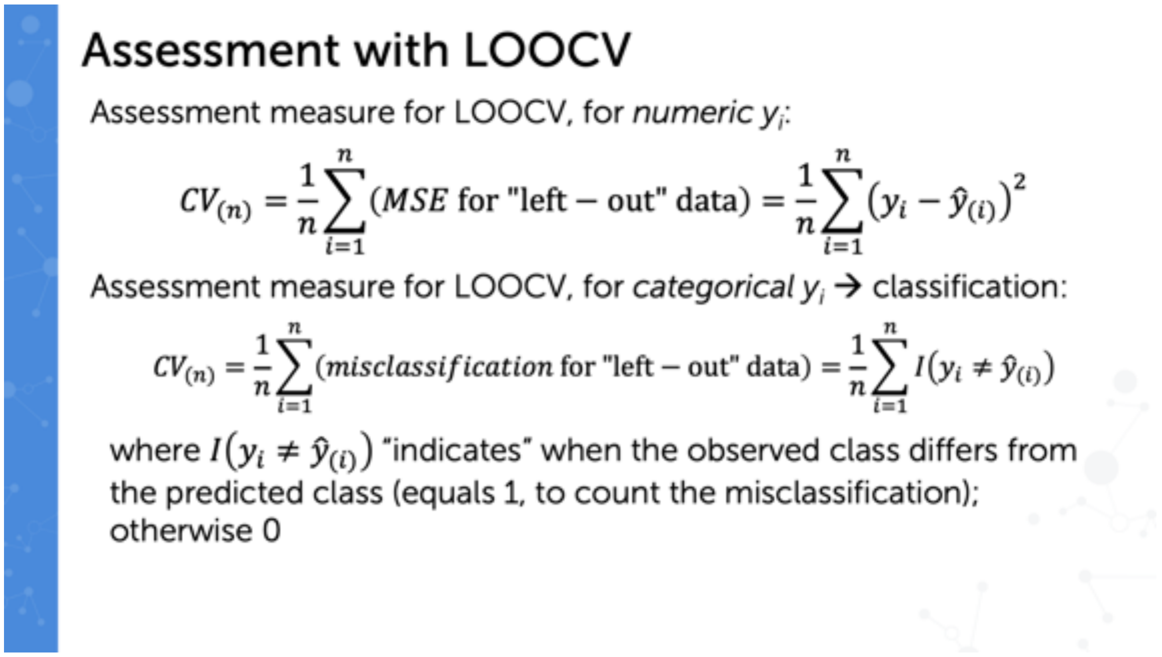
training set 1,2,4 and test set 3 $\rightarrow \hat{y}_{(3)} = \hat{f}_{(3)}(x_3)$

training set 1,2,3 and test set 4 $\rightarrow \hat{y}_{(4)} = \hat{f}_{(4)}(x_4)$

where $\hat{y}_{(i)}$ denotes the prediction for the ith data point based on the model fit on the other (n-1) data points

We now consider Leave One Out Cross-Validation, or LOOCV. This is the simplest application of multiple splits of the data, as we use n to denote number of data points. When we split the data into n sets or folds, we are making splits into subsets of size 1. A very simple example with size n equals 4 illustrates the concept.

We could fit the model on data points 2, 3, and 4. We then use the model to predict data point 1 to get a prediction from a model fit without data point 1, giving us a y hat value. The subscript of 1 in parentheses on the estimated function denotes the prediction from a model fit without data point one. Subsequently leaving out each of the remaining data points, we obtain predictions of each of the data points in an honest way, that is, based on a model fit without that data. Further comparison of these predictions-- y hats-- to the actual observed responses-- y's-- will allow us to assess how well the model can predicts new observations.



Assessment with LOOCV

Assessment measure for LOOCV, for *numeric* y_i:

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n}(MSE \text{ for "left – out" data}) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_{(i)}\right)^2$$

Assessment measure for LOOCV, for *categorical* y_i → classification:

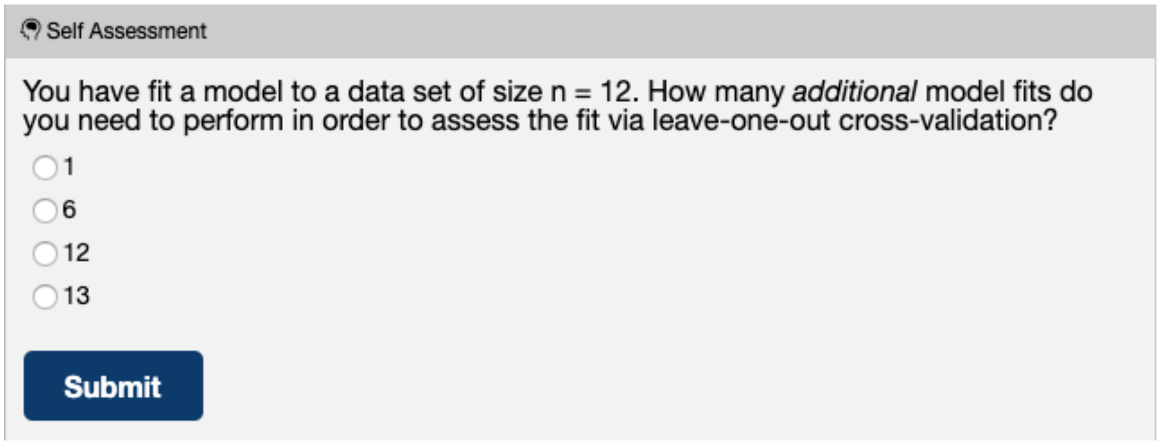
$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n}(misclassification \text{ for "left – out" data}) = \frac{1}{n}\sum_{i=1}^{n} I\left(y_i \neq \hat{y}_{(i)}\right)$$

where $I\left(y_i \neq \hat{y}_{(i)}\right)$ "indicates" when the observed class differs from the predicted class (equals 1, to count the misclassification); otherwise 0

When we have a numeric response, assessment for leave one out cross-validation uses a very similar measure to one which we have already examined. The only distinction is that we denote each predicted data point as having been made from a model fit without the use of that data point by using a subscript i in parentheses to identify the observation that is left out. The formula is as displayed.

The leave one out cross-validation measure for assessment based on these n splits averages out these squared errors over the n splits. This computation is denoted as CV subscript n in parentheses. Note that this formula looks extraordinarily similar to the basic MSE formula with the key difference that the responses are predicted using models fit without that data point.

The cross-validation assessment measures for classification problems are very similar to those using numeric responses, with the only adjustment being that the indicator for misclassification of observations is used in place of the squared error computation. Note that the key difference, again, is that the predictions are performed based on a fitted model using data except for the observation that we are eventually predicting. That is, we have an honest predicted categorization for our new data point.

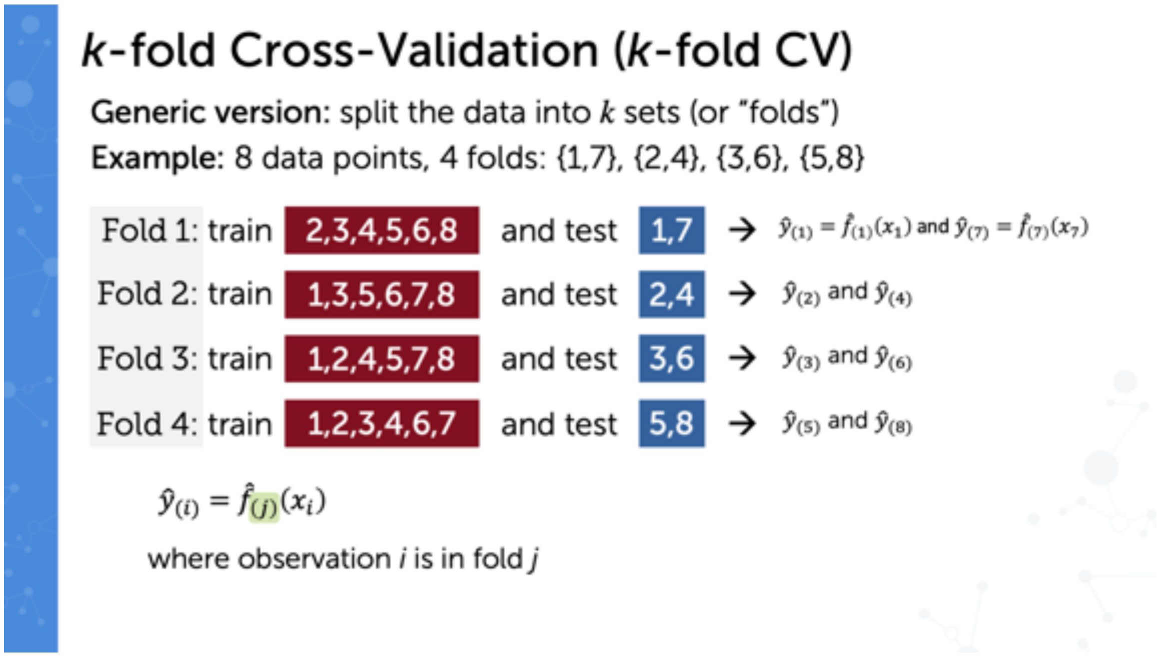


Self Assessment

You have fit a model to a data set of size n = 12. How many *additional* model fits do you need to perform in order to assess the fit via leave-one-out cross-validation?

- ◯ 1
- ◯ 6
- ◯ 12
- ◯ 13

Correct answer is on the last page.


k-fold Cross-Validation (*k*-fold CV)

Generic version: split the data into k sets (or "folds")

Example: 8 data points, 4 folds: {1,7}, {2,4}, {3,6}, {5,8}

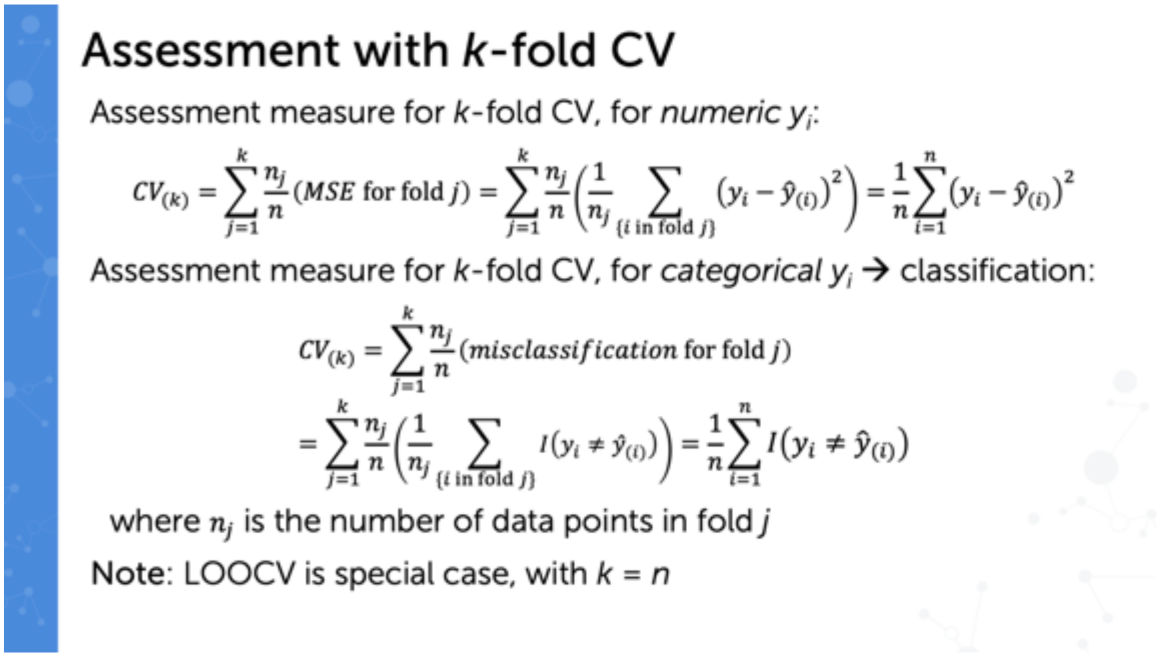
Fold 1: train	2,3,4,5,6,8	and test	1,7	→	$\hat{y}_{(1)} = \hat{f}_{(1)}(x_1)$ and $\hat{y}_{(7)} = \hat{f}_{(7)}(x_7)$
Fold 2: train	1,3,5,6,7,8	and test	2,4	→	$\hat{y}_{(2)}$ and $\hat{y}_{(4)}$
Fold 3: train	1,2,4,5,7,8	and test	3,6	→	$\hat{y}_{(3)}$ and $\hat{y}_{(6)}$
Fold 4: train	1,2,3,4,6,7	and test	5,8	→	$\hat{y}_{(5)}$ and $\hat{y}_{(8)}$

$$\hat{y}_{(i)} = \hat{f}_{(j)}(x_i)$$

where observation i is in fold j

With k fold cross-validation, or k CV, we use k to denote the number of splits. When we split the data into k sets or folds, we are making splits into subsets of approximately the same size, each of which will be n over k. A very simple example of size n equals 8 illustrates the concept. With 4 folds, this means all subsets are of size 2.

We begin by fitting the model on all data points except 1 and 7. We then use the model to predict data points 1 and 7 to get a valid or realistic prediction from a model fit without these data. Subsequently leaving out each of the remaining folds of data, we obtain predictions for each of the data points in an honest way. The subscript in parentheses on the estimated functions still denotes the prediction from a model fit without the data point.



Assessment with *k*-fold CV

Assessment measure for *k*-fold CV, for *numeric* y_i:

$$CV_{(k)} = \sum_{j=1}^{k} \frac{n_j}{n} (MSE \text{ for fold } j) = \sum_{j=1}^{k} \frac{n_j}{n} \left(\frac{1}{n_j} \sum_{\{i \text{ in fold } j\}} (y_i - \hat{y}_{(i)})^2 \right) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_{(i)})^2$$

Assessment measure for *k*-fold CV, for *categorical* y_i → classification:

$$CV_{(k)} = \sum_{j=1}^{k} \frac{n_j}{n} (misclassification \text{ for fold } j)$$

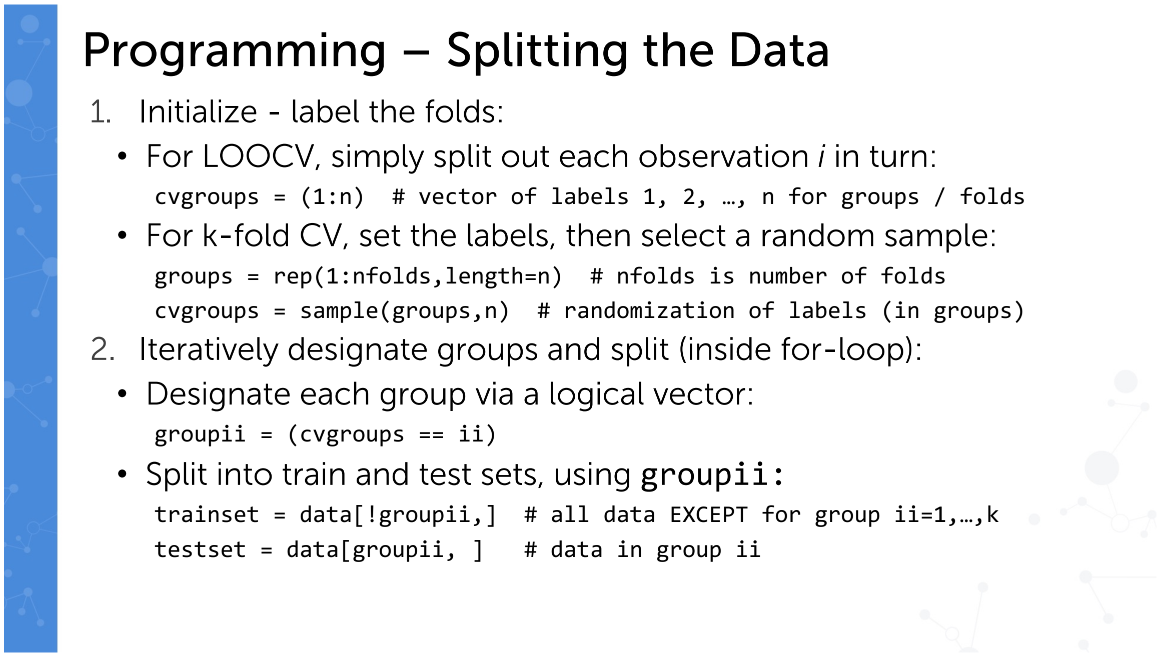
$$= \sum_{j=1}^{k} \frac{n_j}{n} \left(\frac{1}{n_j} \sum_{\{i \text{ in fold } j\}} I(y_i \neq \hat{y}_{(i)}) \right) = \frac{1}{n} \sum_{i=1}^{n} I(y_i \neq \hat{y}_{(i)})$$

where n_j is the number of data points in fold *j*

Note: LOOCV is special case, with *k* = *n*

Assessment for k fold cross-validation uses essentially the same measure as that used for leave one out cross-validation. However, since the folds may be of slightly different sizes, unlike leave one out cross-validation, we use a weighted average of the squared errors computed from within those splits. Thus, the k fold cross-validation measure for assessment based on k splits results in a CV sub k in parentheses.

Note that the final version of the formula looks exactly like the previous one when we work it out to the end. The key difference is that the responses are predicted using models fit without any of the observations within fold j, which is the fold-containing data point i. Conceptually though, once we have the predicted values, the computation of the cross-validation measure for assessment is the same as with leave one out cross-validation. And this applies for both numeric y and categorical y as responses.



Programming – Splitting the Data

1. Initialize - label the folds:
   - For LOOCV, simply split out each observation *i* in turn:

```
cvgroups = (1:n)  # vector of labels 1, 2, …, n for groups / folds
```

   - For k-fold CV, set the labels, then select a random sample:

```
groups = rep(1:nfolds,length=n)  # nfolds is number of folds
cvgroups = sample(groups,n)  # randomization of labels (in groups)
```

2. Iteratively designate groups and split (inside for-loop):
   - Designate each group via a logical vector:

```
groupii = (cvgroups == ii)
```

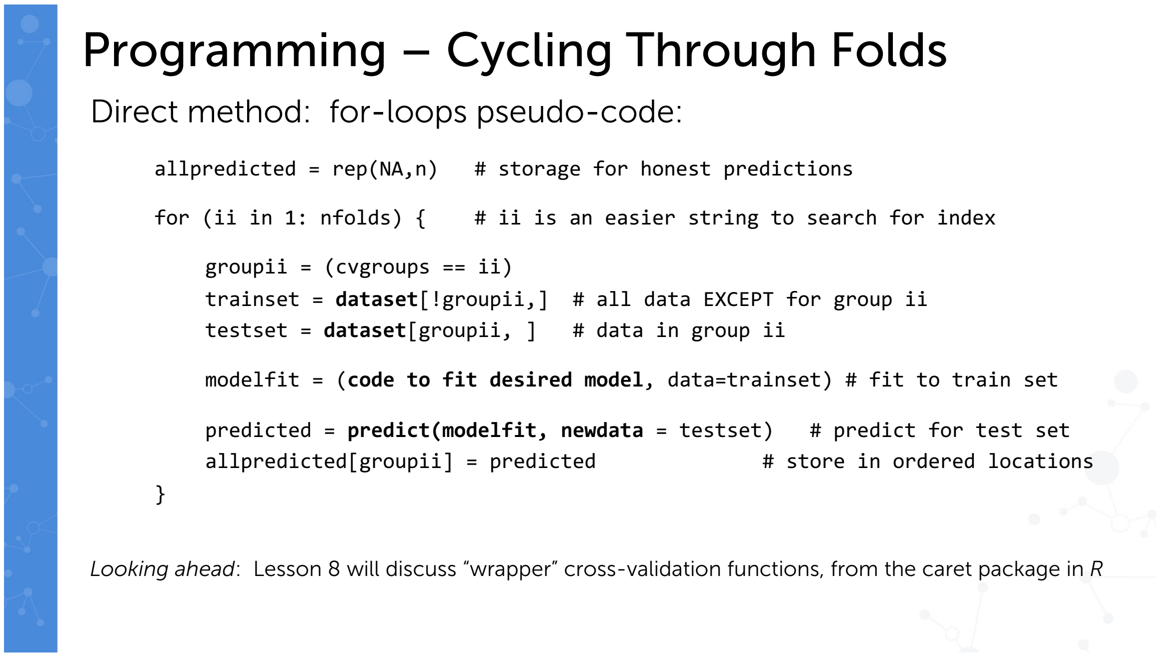
   - Split into train and test sets, using `groupii:`

```
trainset = data[!groupii,]  # all data EXCEPT for group ii=1,…,k
testset = data[groupii, ]   # data in group ii
```

First step in programming cross-validation is obtaining an appropriate split of the data. For leave-one-out cross-validation, this is as simple as cycling through the observations and leaving each out in turn as the test set. Hence, this can be done by simply labeling the cross-validation groups from 1 up to n, leaving each one out in turn, and then using the rest of the data as the training set.

For k-fold cross-validation, however, we need to do a random assignment of labels. So we begin with a set of all possible labels corresponding to the length of the observations, n, that we have in our data set, and labeling with 1 up to number of folds as our labels. We then randomize by using the sample function, and we will do so in conjunction with set.seed typically, to allow for a consistent starting point.

Once we have our cross-validation groups, or stored-in CV groups, we then split the data by iteratively designating each group and using that designation to split into train and test sets. And we'll see this inside a loop on the next slide.



Programming – Cycling Through Folds

Direct method: for-loops pseudo-code:

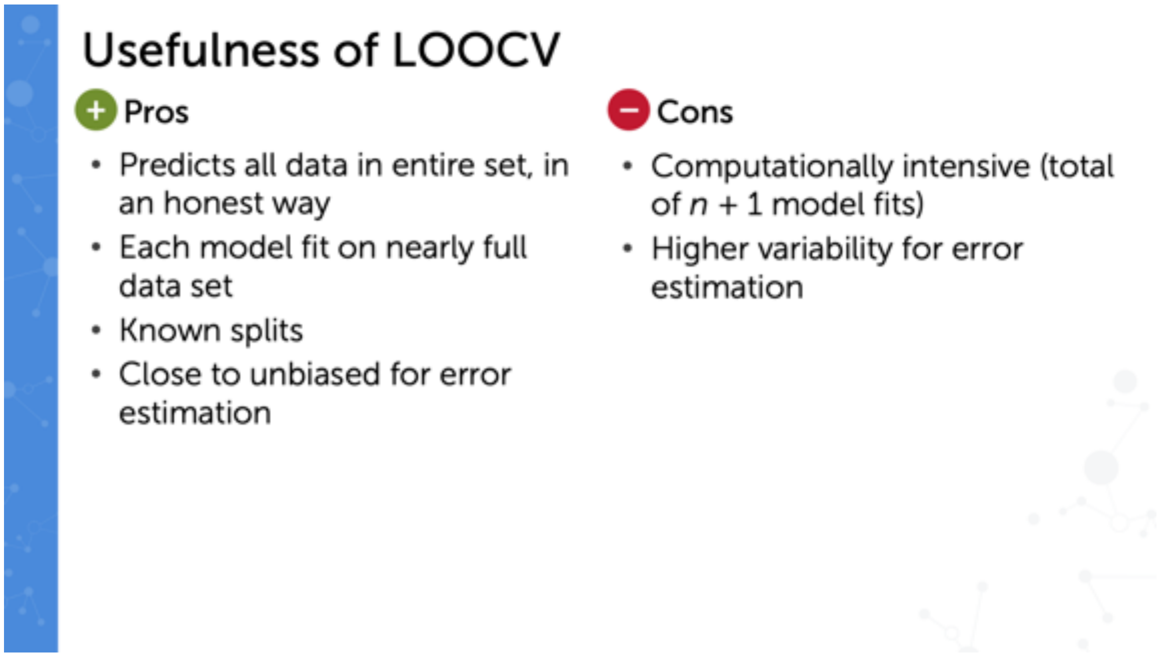
```
allpredicted = rep(NA,n)   # storage for honest predictions

for (ii in 1: nfolds) {    # ii is an easier string to search for index

    groupii = (cvgroups == ii)
    trainset = dataset[!groupii,]  # all data EXCEPT for group ii
    testset = dataset[groupii, ]   # data in group ii

    modelfit = (code to fit desired model, data=trainset) # fit to train set

    predicted = predict(modelfit, newdata = testset)   # predict for test set
    allpredicted[groupii] = predicted              # store in ordered locations
}
```

Looking ahead: Lesson 8 will discuss "wrapper" cross-validation functions, from the caret package in *R*

To integrate the designated splits stored in CV groups, the next step in the programming process requires that we cycle through each fold of data, which we do so using for-loops. Even before starting the for-loop, we need to set up storage space to contain the predicted values from inside each iteration of the loop. We then-- in the next segment of lines of code, the first segment inside the loop-- iterate through designating each group in turn by using a logical vector, CV groups equals some value, and we're going to use II to iterate through the loop since it's an easier search string.

Once we have the logical vector, we use that to designate our train set, not including group I, and our test set, which does include group I data. The next line of code fits the model to whatever train set we've split off. We store that in, say, model fit, and we then use that model fit to predict our test data-- that is, the data in the test set-- and store that in predicted. Since that is just the set of predicted values for this subset of data-- this particular test set-- we then need to further put that into our all-predicted storage in the correct locations as designated buyer, logical vector group I.

A brief comment, looking ahead we'll discuss a wrapper function that does much of this for a wide variety of different modeling techniques, and we'll be talking about that in lesson 8.



Usefulness of LOOCV

Pros

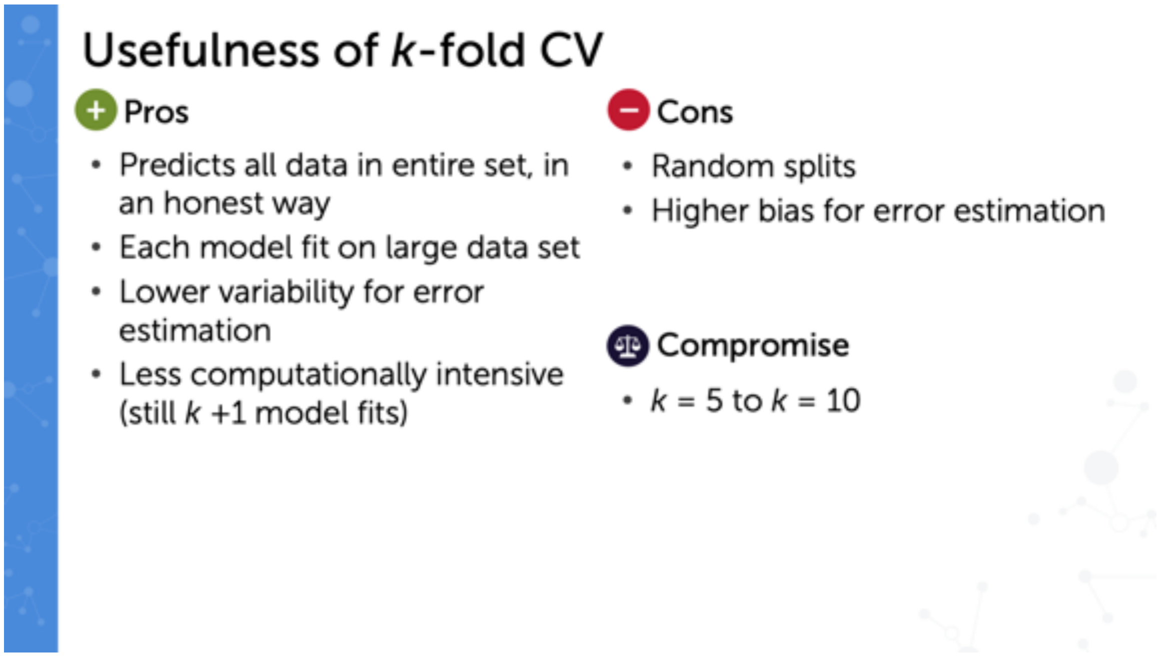
- Predicts all data in entire set, in an honest way
- Each model fit on nearly full data set
- Known splits
- Close to unbiased for error estimation

Cons

- Computationally intensive (total of $n + 1$ model fits)
- Higher variability for error estimation

Leave one out cross-validation is a very useful tool for assessment since it can predict all data in the entire set, yet does so based on models fit without that data. In addition, these models are fit using nearly all the data. The CV measure as an estimate for variability is close to unbiased, and this is the lowest possible bias among all possible choices for splits.

However, the variance of the estimate is increased due to the overlap among the data sets used to fit the models. The splits of the data are simple and deterministic. There is only one possible way to split the data into n folds. This has the downside of making the method computationally intensive with n plus model fits, especially if the model fitting process is time-consuming.



Usefulness of *k*-fold CV

Pros

- Predicts all data in entire set, in an honest way
- Each model fit on large data set
- Lower variability for error estimation
- Less computationally intensive (still k +1 model fits)

Cons

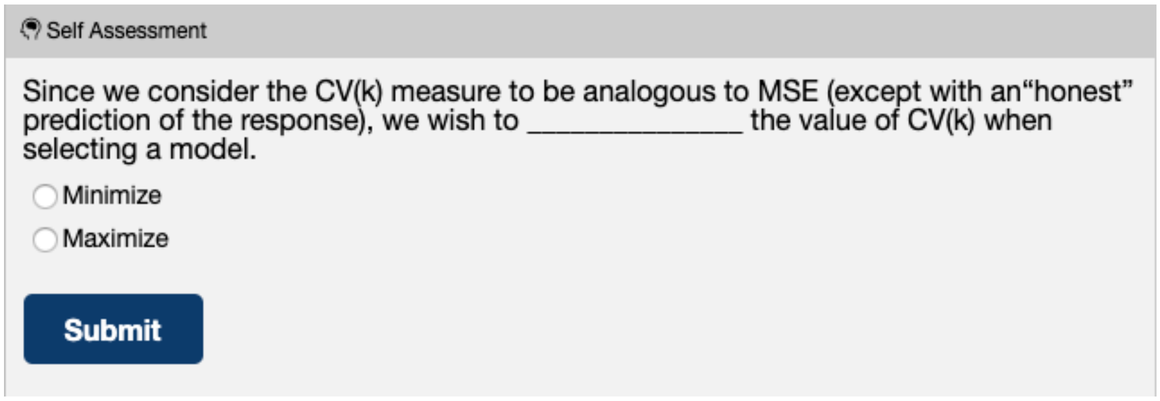
- Random splits
- Higher bias for error estimation

Compromise

- $k = 5$ to $k = 10$

K fold cross-validation, again, predicts all data in the data set based on models fit without that data. In addition, those models are fit using a high proportion of the data. However, since each training set is smaller than with leave one out cross-validation, the CV measure will be slightly biased when estimating the variability. Since there is less overlap among the data sets used to fit the models, the variance is decreased, resulting in a trade-off between bias and variance.

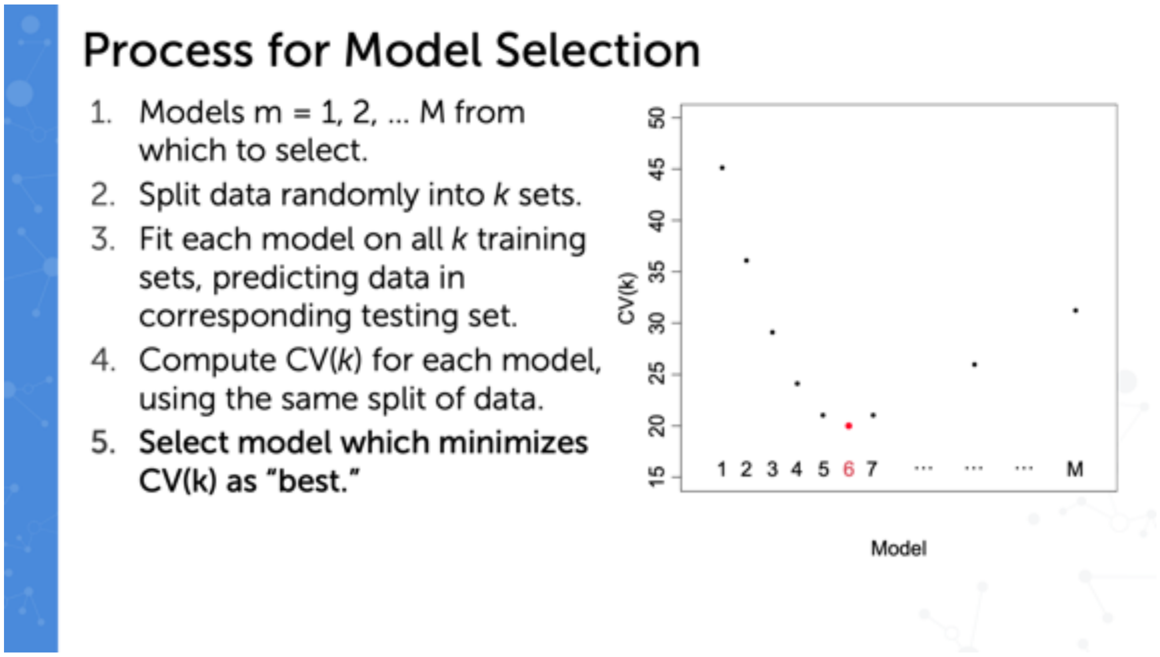
Along with more mathematical evaluations of the cross-validation procedures, this suggests that a good compromise number of splits is a k of something between 5 to 10 folds. The splits of the data are typically done at random. Multiple ways of splitting the data will lead to different CV values. However, this method can be potentially much less computationally intensive with k plus 1 model fits, depending on data size, as well as model-fitting process. This again suggests the use of k fold CV.


Self Assessment

Since we consider the CV(k) measure to be analogous to MSE (except with an“honest” prediction of the response), we wish to _______________ the value of CV(k) when selecting a model.

- ○ Minimize
- ○ Maximize

Correct answer is on the last page.



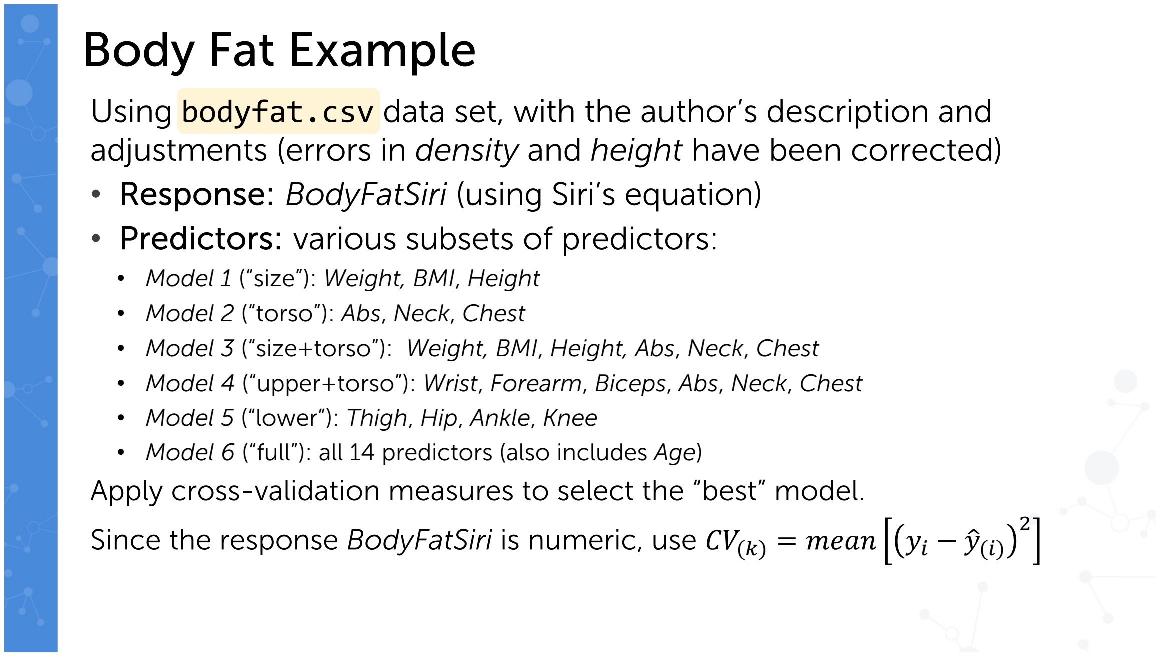
Process for Model Selection

1. Models m = 1, 2, … M from which to select.
2. Split data randomly into *k* sets.
3. Fit each model on all *k* training sets, predicting data in corresponding testing set.
4. Compute CV(*k*) for each model, using the same split of data.
5. **Select model which minimizes CV(k) as "best."**

CV(k)
50
45
40
35
30
25
20
15
1 2 3 4 5 6 7 … … … M
Model

Suppose we have multiple models of which we would like to select the best model for predicting the response. That is, we want to get predictions as close as possible to the true response values. We also want to get honest predictions.

These dual goals suggest using cross-validation and picking the model for which the corresponding measure is minimized. We do this by splitting the data as we typically would for k fold cross-validation, then each model is fit to the training data and applied to the testing data, and a CV k measure is computed for that model. Ideally, a model, or possibly a group of models, will emerge as best. That is, with lowest value of CV k. And we select that as the best model for predicting new data.



Body Fat Example

Using `bodyfat.csv` data set, with the author's description and adjustments (errors in *density* and *height* have been corrected)

- **Response:** *BodyFatSiri* (using Siri's equation)
- **Predictors:** various subsets of predictors:
  - *Model 1* ("size"): *Weight, BMI, Height*
  - *Model 2* ("torso"): *Abs, Neck, Chest*
  - *Model 3* ("size+torso"): *Weight, BMI, Height, Abs, Neck, Chest*
  - *Model 4* ("upper+torso"): *Wrist, Forearm, Biceps, Abs, Neck, Chest*
  - *Model 5* ("lower"): *Thigh, Hip, Ankle, Knee*
  - *Model 6* ("full"): all 14 predictors (also includes *Age*)

Apply cross-validation measures to select the "best" model.

Since the response *BodyFatSiri* is numeric, use $CV_{(k)} = mean\left[\left(y_i - \hat{y}_{(i)}\right)^2\right]$

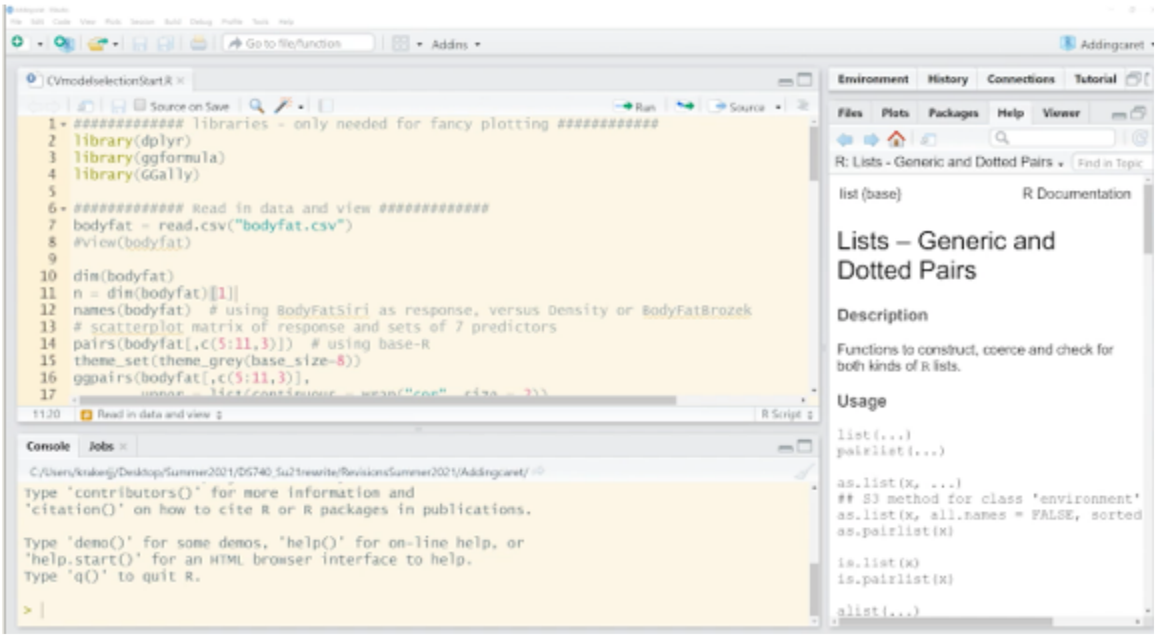
We will be using the body fat data set, with corrections as specified in the reference in the notes from the author. The response is a body fat measurement. There's actually several such measurements in the data set. The one we're going to be using is body fat Siri. It's computed using measurements via a particular equation called series equation.

The predictors that we have available are a set of 14 different variables-- which are very easy-to-measure physical characteristics-- and regroup these by location on the body as a way of talking about different sets or subsets of measurements that might be the most meaningful.

So we will look at five different models-- that include subsets of the variables, and one model that includes all 14 possible predictor variables. We will then apply cross-validation measures to select which of these is the best model, using a CV measure that is effectively our honest MSE-- or honest mean square error, since our response is numeric.

Notes:
The bodyfat.csv file is available in the online course.

The data set author's description and adjustments is available here:
http://www.amstat.org/publications/jse/v4n1/datasets.johnson.html



```r
############# libraries - only needed for fancy plotting ############
library(dplyr)
library(ggformula)
library(GGally)

############# Read in data and view #############
bodyfat = read.csv("bodyfat.csv")
#View(bodyfat)

dim(bodyfat)
n = dim(bodyfat)[1]
names(bodyfat)  # using BodyFatSiri as response, versus Density or BodyFatBrozek
# scatterplot matrix of response and sets of 7 predictors
pairs(bodyfat[,c(5:11,3)])  # using base-R
theme_set(theme_grey(base_size=8))
ggpairs(bodyfat[,c(5:11,3)],
        upper = list(continuous = wrap("cor", size = 3))
```

```
C:/Users/krakeg/Desktop/Summer2021/DS740_Su21rewrite/RevisionsSummer2021/Addingcaret/
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 
```

R: Lists - Generic and Dotted Pairs

list (base) R Documentation

Lists – Generic and Dotted Pairs

Description

Functions to construct, coerce and check for both kinds of R lists.

Usage

```r
list(...)
pairlist(...)

as.list(x, ...)
## S3 method for class 'environment'
as.list(x, all.names = FALSE, sorted
as.pairlist(x)

is.list(x)
is.pairlist(x)

alist(...)
```

This slide represents a video/screencast in the lecture. The transcript does not substitute video content.

We practice the application of cross-validation for model selection using the body fat data set as briefly introduced on the previous slide. We use a first sequence of R commands to read the data set into R and review the data. We'll begin with a few libraries, and these libraries are really only necessary if you are going to be doing some of the fancier plotting later on.

I'm going to make sure that I've got my working directory set to the correct location, and then take a look at the body fat data. So I'm going to take a look at the size of the data set, as well as the names. I could also open up the data set if I felt so inclined in a separate tab. Although that line is commented out because this is not an essential part, but it can help you take a look at the type of data you have, specifically that all the values in this data set are given or numeric.

Also, we're going to be using just one of the variables, body fat siri, as our preferred variable. We'll have case-- or as our preferred response, we'll have case, which is not a used variable. Another body fat and density measures, body fat brozek and density, which will not be used, because there are very close analogies of body fat siri.

Instead, what we're going to be using are some subset of the remaining variables, age, body size, weight, height, BMI, some upper body measurements, some lower body measurements, basically, body measurements that are very easy to obtain. And we're going to take a look, and we could use just base R to view our data. If we did that, we're going to get a readable, but not very visually appealing description of how body fat relates to the different variables.

Now, you'll notice that I organized or arranged the variables-- the order of the variables so that the response is listed last, and what that does in a scatterplot matrix is it ensures that the response is listed on the vertical axis, which is how we think about it, and the potential predictors on the horizontal axis. We see that a lot of these predictors, at least among the first 6 or 7 or so of them are strongly correlated to body fat siri but are also associated with each other, and that might suggest just being able to use subsets of them.

More visually appealing-- visually appealing scatterplot matrix starts with setting the size of the text to be smaller so that it shows up on screen here well enough and that plots the same data but in the upper quadrant is going to show the correlation at a size that is going to be visually legible. And so this one takes a little bit longer to process because it's a more complex visual. So we'll take a look at that one.

Again, it'll be some of the similar observations. That is body fat siri on the vertical axis down here. The various predictors, age, weight, height, BMI on the horizontal axis of each plot. This also provides a smooth curve-- a smooth type histogram of each of the variable values as well as lists the correlation in this upper quadrant.

We could do something similar for the last 7 variables. The last 7 potential predictors, and you're welcome to take a look at that on your own. It gives a very similar type of plot.

So we have used these first 21 lines or so, again, depending on whether or not you use the more complex visuals. It could be fewer lines than that just to get a feel for the data. We then specified the linear models we're going to be fitting. The body fat siri modeled on three variables, size, effectively, variables, weight, BMI, height, and we're going to store that. Call that model 1.

Similarly, we had the torso measurements, the torso and body size measurements, the upper body measurements, the lower body measurements, and then all the predictors. Now, this is a model specification that's a little bit more succinct than simply taking the response modeled on all 14 predictors. We could certainly do that, but it's more succinct to say, I want to model this on the rest of the predictors except for case, except for body fat brozek, except for density.

So now that we have those models specified, we want to apply cross-validation to get an honest measure of the model's predictive ability. And in order to set up cross-validation, we need to define labels for cross validation groups. If we were performing leave one out cross-validation, we would define the number of folds to be n and the CV groups to simply be the integers 1 up to n.

In this case, instead, I'm going to be using 10-fold cross validation, because that is perhaps the slightly more difficult application of this. So I'm going to define n folds equals 10. I'm going to set a seed, and that seed will be required so that you produce the same random sample eventually that I do using the sample function. So I'm going to run those two lines, then I'm going to set up my groups as my repeat. My fold labels up to as many times as I need to.

And if we take a look at what's inside groups, it does a whole bunch of repeats of the digits 1 through 10 up through 252 total observations. And finally, CV groups randomly reorders them. Well, again, randomly to produce the same random reorder, we use a set seed to get consistent results so that you can follow this example and it produces the exact same thing that we will see on screen here.

So if you typed in CV groups and you increased this display area from the console, you should see this exact same listing of CV groups. And we're going to be using those CV groups to run our cross-validation process for just one of the models. Now, how we do that is-- well, I'm going to copy directly from the slide in which we saw for loops pseudocode, and we're going to revise that here to include a model fit. We're going to use model 1 in this case.

So the point for doing-- the purpose for doing this is to help you see that we are actually just doing the code with slight adjustments. Now, I mentioned that I copied this all predicted rep NA n. That's my setup for the storage. I already had that on screen, so we don't need to do that twice, and what I want to do next is loop through with index double ii.

I'm just going to refer to that as i. So my index i going from 1 to n folds. I'm going to define a logical vector. And if you like, you can add in a comment specifying that. Logical vector for group ii, and we then use that logical vector not. So that is the negation of the truth and forces in there to define our training set.

Now, here's the first place where we have to make sure that we've referring to the correct data set. So the data that we're using here is called body fat data set, and there's a couple of ways we could handle this. If I want it to be a little bit bigger picture and reuse the exact same code, I could say data set equals body fat and then run this code as is.

In this case, I'm actually just going to, instead of data set, type in body fat, which is my data set. Only selecting the rows that are not in group i, and then for my test data, I'm going to select just the rows that are in group i. And so these brackets specify rows and columns, and we are just subsetting the rows and the observations. So those first three lines are ready to go.

Now, I see a little x here, because I don't have an actual model specification. I have to type that in, and the type of model that I'm fitting is an LM for linear model. And you'll notice that I had previously defined by my models up here, so I can just refer to them by their labels. The model specification in model 1.

The data is going to be my train set data, and then when I make predictions, and it should work to apply the generic predict function here on my model fit using new data test set. This line will have to be adjusted for certain applications or applications of certain methods, depending on how predict is integrated with that method. I'll then take those values that are stored in predicted and store them in all my predicted values but only in locations where group i is true.

Now, good practice is to always try an application of this loop before you actually run the full loop. So I'm going to set ii equals 1 and run these commands. So notice that I'm not actually doing the four part, I'm just running each inner line. Whoops, except I forgot to write to run my ii line.

I'm going to run each inner line at a time. Now, everything seems to run properly up until this point because I forgot to define my all predicted. Let's see. Wouldn't hurt to just take a look at everything that's inside here. Group i is a vector of true falses like I expected. Check out the dimensions of my train set, the dimension of my test set, and so on and so forth.

So internal to the loop, you can check out whether each line is operating as you intended it to. You can also do a summary of models fit, which would have been fit only to the training set of 226 observations, and of course, that seems to display a summary of a linear model. I could take a look at the 26 observations that should be stored and predicted. And now, when I put them in only the group i locations, when I type in all predicted, I should only have 26 of the 252 locations filled in with values, because I have only done the first loop here.

All right, so now that I verified that those inner workings are running correctly, let me run the full loop. It doesn't take long at all. I'm going to verify that all predicted actually contains values in all 252 locations, and then I would like to compare all honestly predicted values to my response. And so for simplicity in the coding here, I'm just going to relabel this as y.

My y is my full set of response values from the body fat data set. I'm going to compute my CV value as the mean of the squared errors using honest predicted values and display that. 34.49. That in and of itself doesn't have an intrinsic meaning, but I'm going to use that to compare across separate models.

And if I put the response and predicted values together, I can use some visualizations from GG formula, or I could make a simpler plot. I'll add that in here as a and then comment it out. I could make a simpler plot that just takes y hat y with the various titles and labels. So this would be the base R application after, and I will need to correct and put this in as all predicted because I'm not operating from my data frame.

Or if I want to operate from this data frame, that includes my observed and predicted values called y and y hat, respectively, I can make a more palatable visual. The last two parts of this asks us to add some efficiency. And so if I wanted to redo this process, for my different models, I would have to go in and change my model label. I would also have to go in and change my storage.

So this could be a little bit meticulous if I'm trying to go through and do this model after model. So an added efficiency could be to simply put the models together in a list, and it might also be helpful to make a list to store my all CV values. And then I'm going to pick out one of the models, say, the first model and run the loop across that.

So it's going to be very similar to this process up above. In fact, I'm going to copy all the commands over here, except now, all I have to change is the value of m. And I can do this instead of referring to the name of the model by referring to the m-th location in all models. And when I come down here for CV value, instead of calling the CV value 1 and then having to relabel that, I can instead just store this in a location on all CV values.

So all CV values location m is going to be the mean of the squared errors. And finally, I can change the titles a bit by simply pasting to the end of the designation of the name or of the title. And so this will add a little bit of streamlining. So let's see this.

If I repeat this process now for m equals 1, and my mistake was that all models here is actually a list, not a vector, and so I have to specify the list item m. Now that operates as planned, and I can see that it went through the same process. The cross validation process using all models one, the first model in my list of models, and then stores my information.

And if I wanted to get base R to operate correctly, I need to change that to mean. And my fancier plotting looks similarly shaped, just a little bit more visually [INAUDIBLE]. OK, so great. So we streamlined-- or so we made that process a little bit more efficient in that we only had to change m here at the beginning.

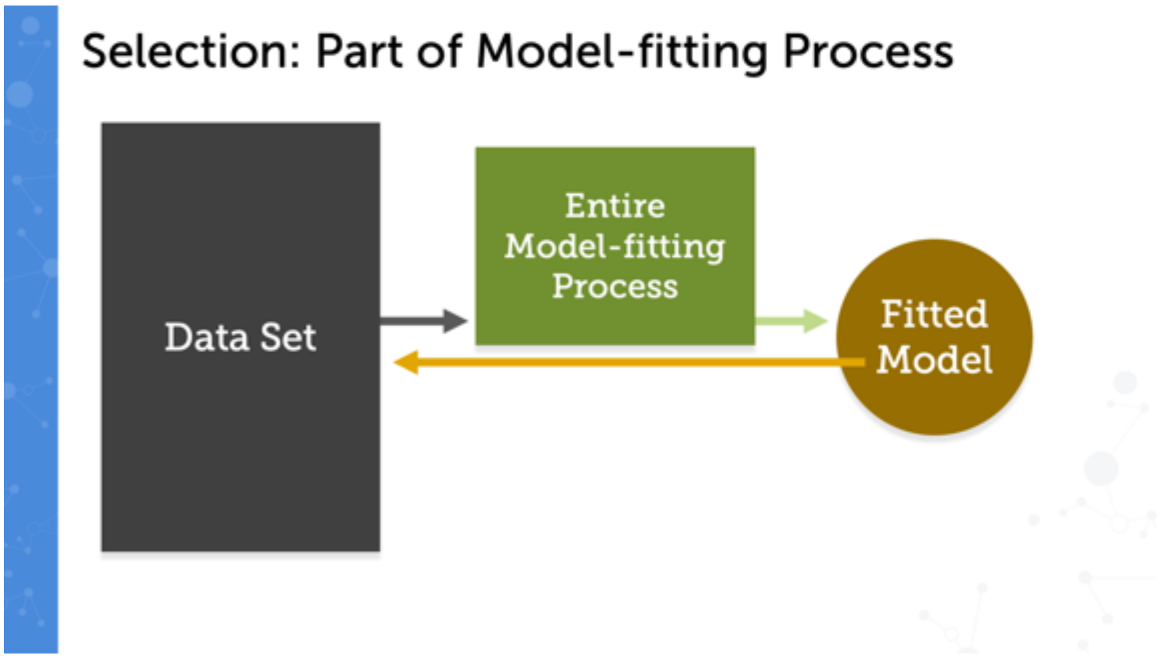
So one final step that could add some streamlining to this is to make a for loop to cycle through the models. And so I am going to open up a document, where I have done this and just copied that over. And you can see I now have CV model selection open, and so that I don't have to retype all of this.

We're taking realistically the entire process from up here that we applied for a particular value of m. We set up storage, we ran a loop to do the cross validation, and we stored the value for our CV as one of the items in all CV values. And so we take those steps, set up our storage, run the cross validation loop, compute CV and store it, and just getting a little extra fancy here, I am going to store in a list of all plots, I am going to store the plot for model m.

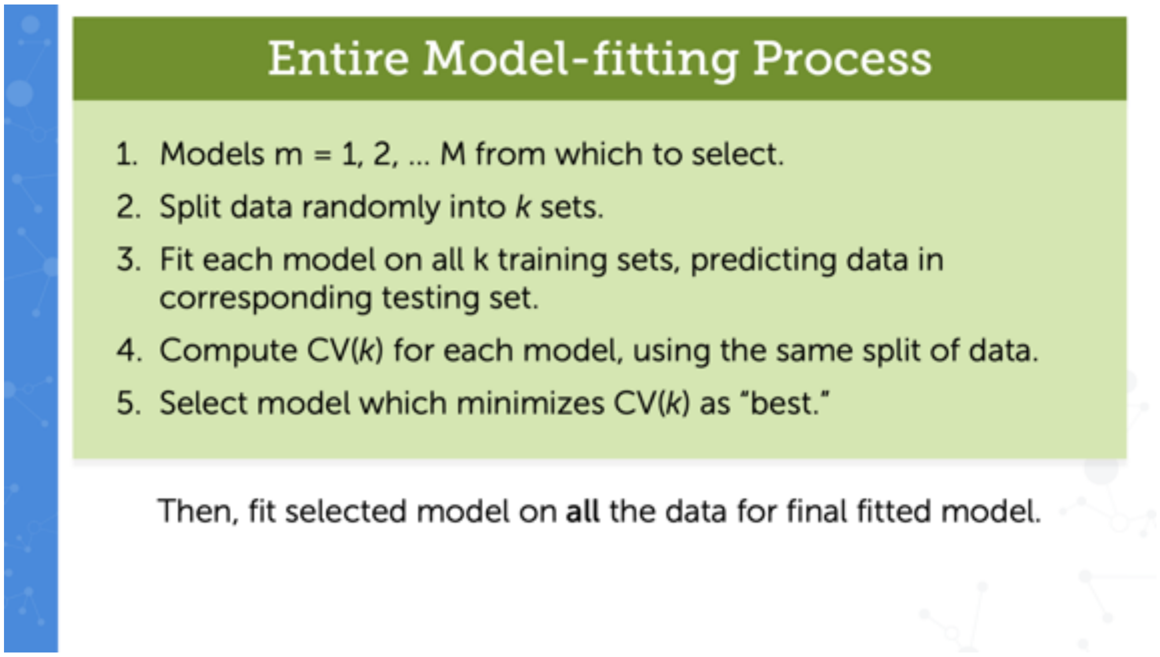
So if I set up a list, and then I run this across all models, now, I know that this internal stuff works already because I practiced that up here for m equals 1. And so for m, cycling through all my different models, it's going to run all those interior cross validations. At the end of this, I will get all CV values for models 1 through 6, and I will see that the model that appears to have minimized is model 3. Has the minimum CV value.

And so in fact, I am going to want to take a look at the plot of the observed minus predicted for that model 3. Is best. And several of the others are close in terms of CV values, but model 3 really does the best job. If we wanted to compare that to a model that really doesn't do a very good job in predictions corresponding to the observed values, we could take a look at all plots 5.

So all plots for model 5, and we see what happens here is that there are some outlying values that where the predictions do not match up very well with the observed, and that inflates the mean square. So hopefully this helps you build your own loop for cross-validation and demonstrates how, indeed, we can use that for model selection.


Selection: Part of Model-fitting Process
Data Set
Entire Model-fitting Process
Fitted Model

It is now of paramount importance to recognize that this model selection step has been incorporated as part of the model-fitting process. That is, to get from the data set to the fitted model, we must proceed through all steps of the model-fitting process.

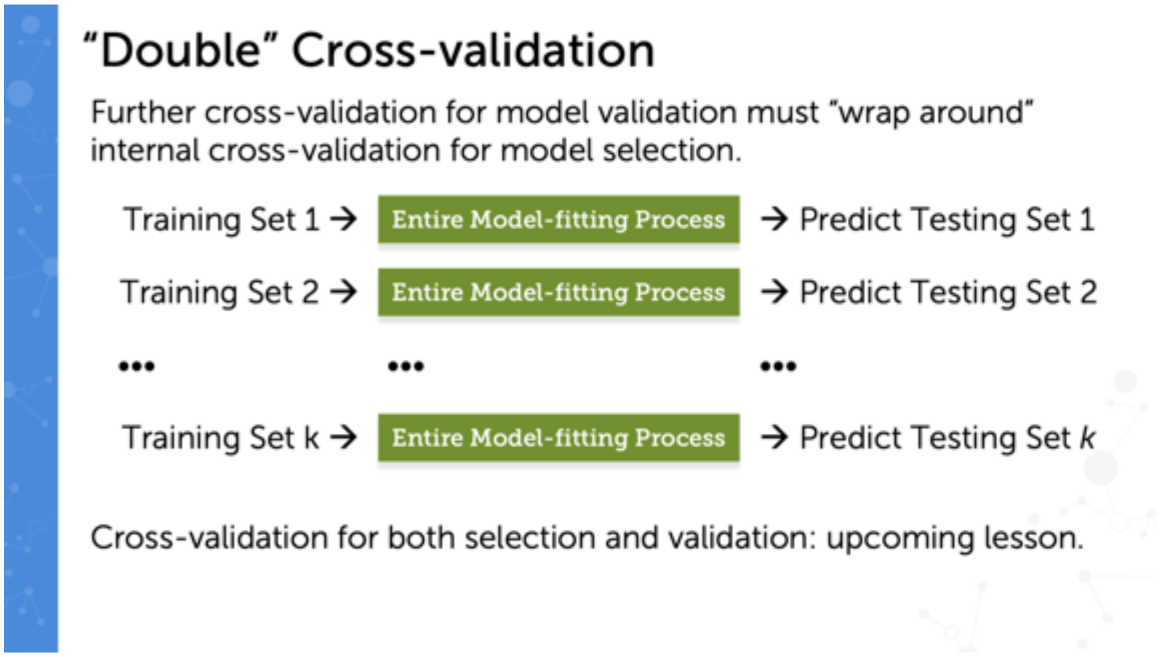


Entire Model-fitting Process

1. Models m = 1, 2, ... M from which to select.
2. Split data randomly into *k* sets.
3. Fit each model on all k training sets, predicting data in corresponding testing set.
4. Compute CV(*k*) for each model, using the same split of data.
5. Select model which minimizes CV(*k*) as "best."

Then, fit selected model on **all** the data for final fitted model.

Since cross-validation is used to select the model, it has been incorporated as part of the model-fitting process. When fitting a model, we must proceed through all steps that were used in the process, including this cross-validation for model selection. That means every time we select a model, we must implement a cross-validation. Why is this important?



"Double" Cross-validation

Further cross-validation for model validation must "wrap around" internal cross-validation for model selection.

Training Set 1 → Entire Model-fitting Process → Predict Testing Set 1

Training Set 2 → Entire Model-fitting Process → Predict Testing Set 2

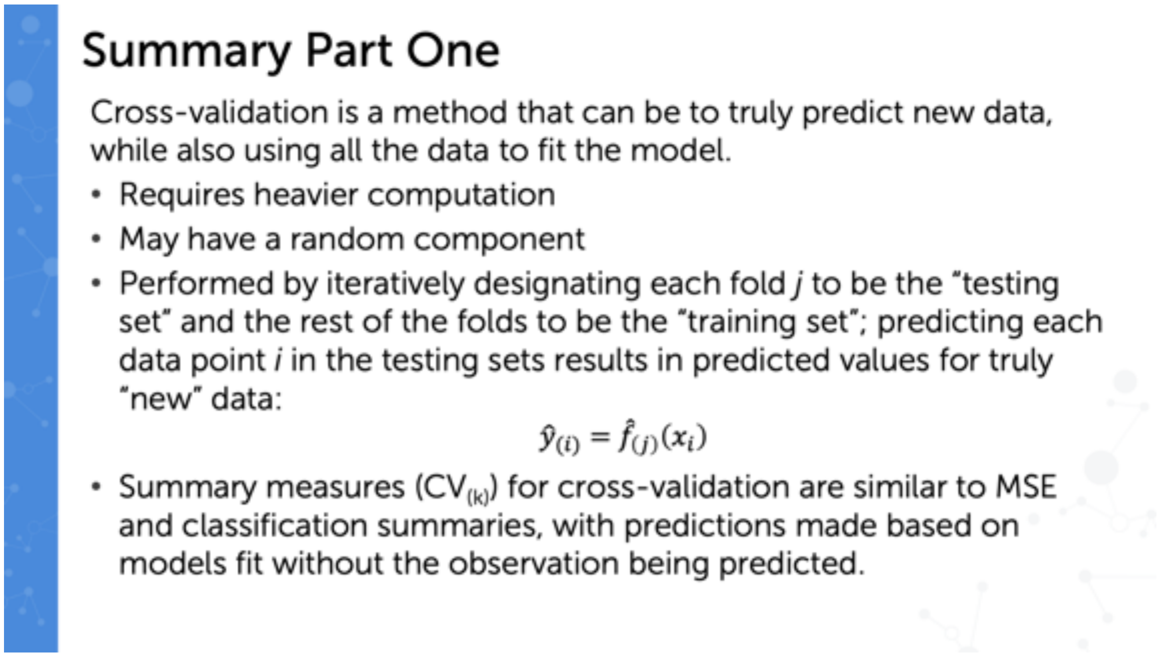
••• ••• •••

Training Set k → Entire Model-fitting Process → Predict Testing Set *k*

Cross-validation for both selection and validation: upcoming lesson.

This leads to what is known as the double application of cross-validation. If we are to use cross-validation for the model selection process, we need to recognize that there will be another level of cross-validation surrounding that used to validate the overall predictive ability of the model. That is, we will eventually be using two layers of cross-validation. The outer split of the data is used for model validation, while the inner split is used for model selection.

As this is a more complex application of the methods, we will wait until further applications and lessons to implement this double usage of cross-validation. So for now, in the next several lessons, we will first focus on applying cross-validation simply for model selection. And we will practice that through a variety of models.



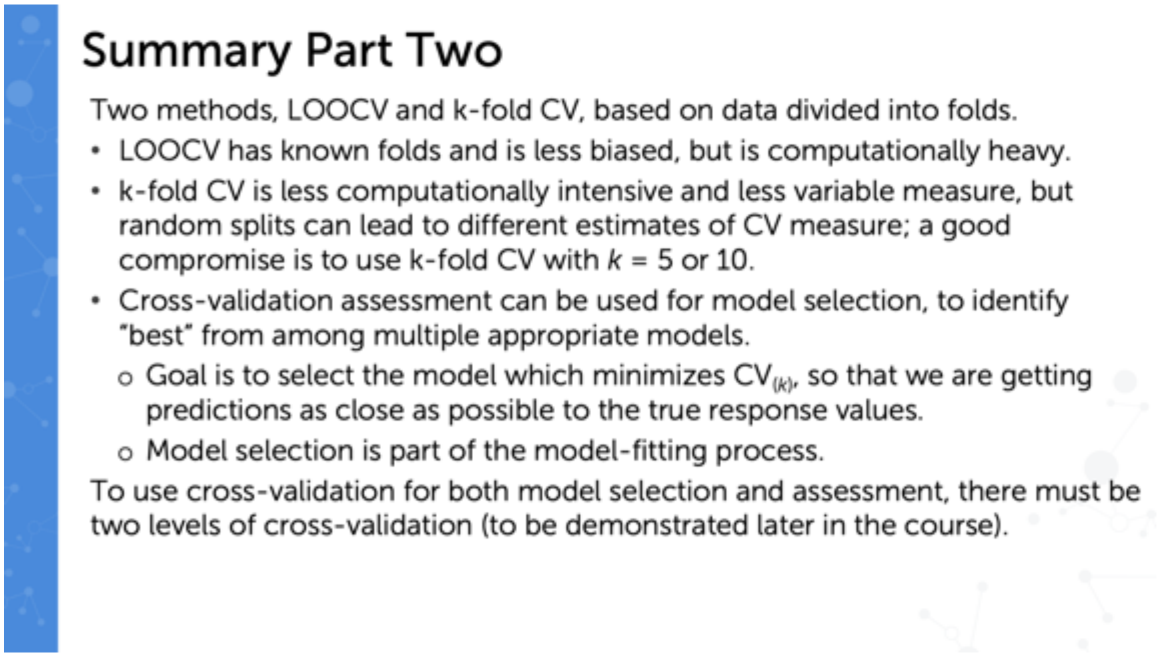
Summary Part One

Cross-validation is a method that can be to truly predict new data, while also using all the data to fit the model.

- Requires heavier computation
- May have a random component
- Performed by iteratively designating each fold *j* to be the "testing set" and the rest of the folds to be the "training set"; predicting each data point *i* in the testing sets results in predicted values for truly "new" data:

$$\hat{y}_{(i)} = \hat{f}_{(j)}(x_i)$$

- Summary measures ($CV_{(k)}$) for cross-validation are similar to MSE and classification summaries, with predictions made based on models fit without the observation being predicted.

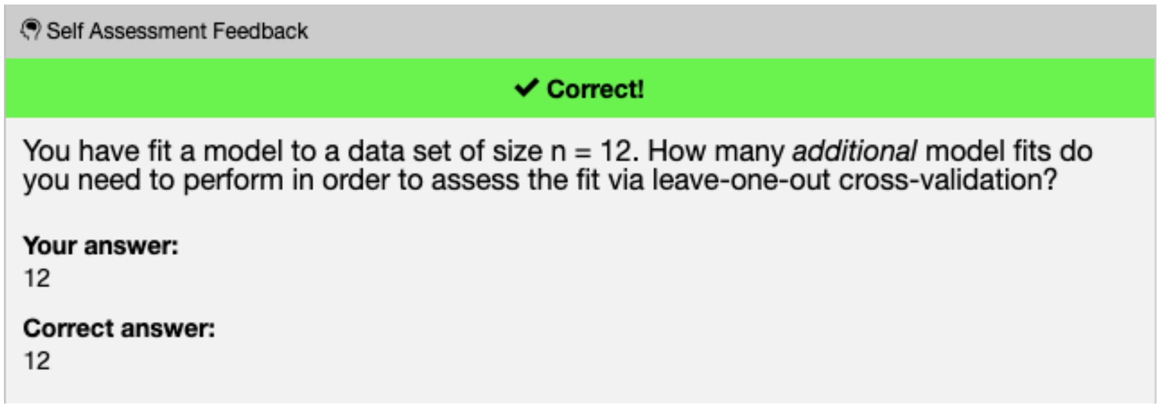


Summary Part Two

Two methods, LOOCV and k-fold CV, based on data divided into folds.

- LOOCV has known folds and is less biased, but is computationally heavy.
- k-fold CV is less computationally intensive and less variable measure, but random splits can lead to different estimates of CV measure; a good compromise is to use k-fold CV with $k = 5$ or 10.
- Cross-validation assessment can be used for model selection, to identify "best" from among multiple appropriate models.
  - Goal is to select the model which minimizes $CV_{(k)}$, so that we are getting predictions as close as possible to the true response values.
  - Model selection is part of the model-fitting process.

To use cross-validation for both model selection and assessment, there must be two levels of cross-validation (to be demonstrated later in the course).

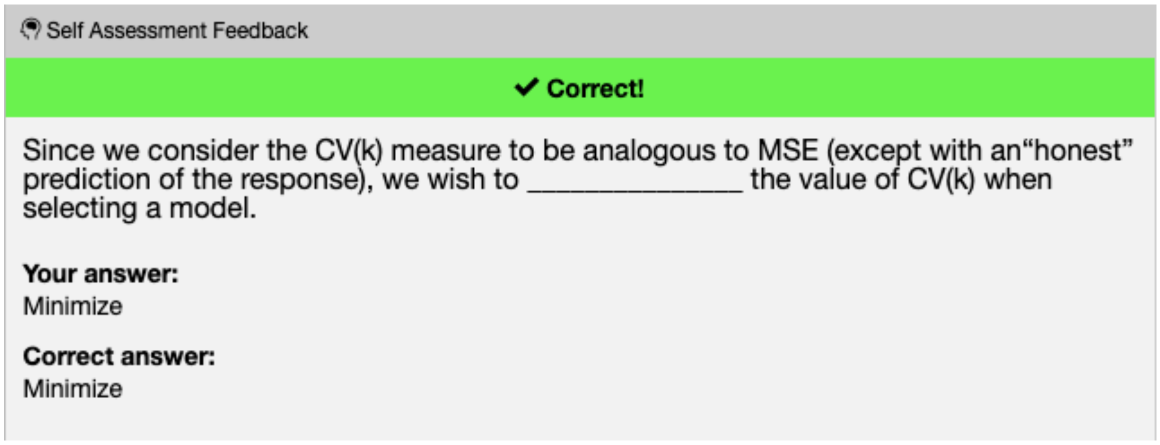

Self Assessment Feedback

✔ Correct!

You have fit a model to a data set of size n = 12. How many *additional* model fits do you need to perform in order to assess the fit via leave-one-out cross-validation?

Your answer:
12

Correct answer:
12


Self Assessment Feedback

✔ Correct!

Since we consider the CV(k) measure to be analogous to MSE (except with an“honest” prediction of the response), we wish to _______________ the value of CV(k) when selecting a model.

Your answer:
Minimize

Correct answer:
Minimize