Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *doi: 10.18637/jss.v000.i00*

cv: An R Package for Cross-Validating Regression Models

John Fox
McMaster University

Georges Monette
York University

Abstract

We describe the **cv** package, which implements cross-validation for standard R regression models through a uniform and simple to use generic function. Methods for the `cv()` function are provided for many commonly employed classes of statistical models, including mixed-effects models, and it is straightforward to extend the **cv** package by writing methods for additional classes of models. The `cv()` function also can cross-validate complex model-selection procedures, such as those that include variable transformations, predictor selection, and selection among competing models. The `cv()` function generally supports parallel computations, and the supplied methods for linear and generalized linear models take advantage of computational efficiencies.

Keywords: cross-validation, regression analysis, model selection, R.

1. Introduction

Cross-validation (CV) is an essentially simple and intuitively reasonable approach to estimating the predictive accuracy of regression models. CV is developed in many standard sources on regression modeling and "machine learning"—we particularly recommend James, Witten, Hastie, and Tibshirani (2021, Secs. 5.1, 5.3)—and so we will describe the method only briefly here before taking up computational issues and some examples. See Arlot and Celisse (2010) for a wide-ranging, if technical, survey of cross-validation and related methods that emphasizes the statistical properties of CV.

Validating research by replication on independently collected data is a common scientific norm. Emulating this process in a single study by data-division is less common: The data are randomly divided into two, possibly equal-size, parts; the first part is used to develop and fit a statistical model; and then the second part is used to assess the adequacy of the model fit to the first part of the data. Data-division, however, suffers from two problems: (1)

Dividing the data decreases the sample size and thus increases sampling error; and (2), even more disconcertingly, the results can vary substantially based on the random division of the data, particularly in smaller samples. See Harrell (2015, Sec. 5.3) for this and other remarks about data-division and cross-validation.

Cross-validation speaks to both of these issues. In CV, the data are randomly divided as equally as possible into several, say k, parts, called "folds." The statistical model is fit k times, leaving each fold out in turn. Each fitted model is then used to predict the response variable for the cases in the omitted fold. A CV criterion (also termed a "cost" or "loss" measure), such as the mean-squared error ("MSE") of prediction, is then computed using these predicted values. In the extreme $k = n$, the number of cases in the data, thus omitting individual cases and refitting the model n times—a procedure termed "leave-one-out (LOO) cross-validation."

Because the n models are each fit to $n-1$ cases, LOO CV produces a nearly unbiased estimate of prediction error. The n regression models are highly statistical dependent, however, based as they are on nearly the same data, and so the resulting estimate of prediction error has relatively large variance. In contrast, estimated prediction error for k-fold CV with $k = 5$ or 10 (commonly employed choices) are somewhat biased but have smaller variance. It is also possible to correct k-fold CV for bias (see Section 7.2).

The **cv** package for R automates the process of cross-validation for standard R statistical model objects. The principal function in the packages, also named `cv()`, has methods for objects produced by a number of commonly employed regression-modeling functions, including those for mixed-effects models:

```
R> library("cv", quietly=TRUE)
R> methods("cv")

[1] cv.default*  cv.function* cv.glm*      cv.glmmTMB*  cv.lm*
[6] cv.lme*      cv.merMod*   cv.modList*  cv.rlm*
see '?methods' for accessing help and source code
```

- The `"lm"` and `"glm"` methods are for linear and generalized-linear models fit respectively by the standard R `lm()` and `glm()` functions.
- The `"modList"` method for `cv()` cross-validates several competing models, not necessarily of the same class, using the same division of the data into folds. The `cv()` function is introduced in the context of a preliminary example in Section 2 of the paper.
- Cross-validating mixed-effects models, using the `"glmmTMB"`, `"lme"`, and `"merMod"` methods for `cv()`, involves special considerations that we take up in Section 3.
- The `"function"` method for `cv()`, discussed in Section 4, cross-validates a complex model-specification process that may, for example, involve choice of data transformations and predictors, or model selection via CV itself.
- The `"default"` `cv()` method works (perhaps with a bit of coaxing) with many other existing regression-model classes for which there is an `update()` method that accepts a `data` argument. More generally, the **cv** package is designed to be extensible, as discussed in Section 5.

A number of existing R packages include functions for cross-validating regression models. We briefly situate the **cv** package relative to other R software for cross-validation in Section 6.

The final section of the paper (Section 7) describes some computational details, including efficient CV computations for linear and generalized-linear models and computation of bias-corrected CV criteria.

In the interest of brevity, we won't describe all of the features of the **cv** package here, concentrating on the aspects of the package that are relatively novel. For example, the `cv()` function can perform computations in parallel and can independently replicate a cross-validation procedure several times. There are also data management facilities in the package, such as coercing the objects produced by the `cv()` function into data frames for further analysis. These and other features not discussed in this paper are taken up in the vignettes distributed with the package, which also provides greater detail on some of topics that we do describe, such as extensions to the package.

2. Preliminary Example: Polynomial regression

The data for the example in this section are drawn from the **ISLR2** package for R, associated with James *et al.* (2021). The presentation here is close (though not identical) to that in the original source (James *et al.* 2021, Secs. 5.1, 5.3), and it demonstrates the use of the `cv()` function.[1]

The `Auto` dataset contains information about 392 cars:

```
R> data("Auto", package="ISLR2")
R> summary(Auto)
```

```
      mpg          cylinders      displacement     horsepower        weight
 Min.   : 9.00   Min.   :3.000   Min.   : 68.0   Min.   : 46.0   Min.   :1613
 1st Qu.:17.00   1st Qu.:4.000   1st Qu.:105.0   1st Qu.: 75.0   1st Qu.:2225
 Median :22.75   Median :4.000   Median :151.0   Median : 93.5   Median :2804
 Mean   :23.45   Mean   :5.472   Mean   :194.4   Mean   :104.5   Mean   :2978
 3rd Qu.:29.00   3rd Qu.:8.000   3rd Qu.:275.8   3rd Qu.:126.0   3rd Qu.:3615
 Max.   :46.60   Max.   :8.000   Max.   :455.0   Max.   :230.0   Max.   :5140

  acceleration        year           origin                      name
 Min.   : 8.00   Min.   :70.00   Min.   :1.000   amc matador       :  5
 1st Qu.:13.78   1st Qu.:73.00   1st Qu.:1.000   ford pinto        :  5
 Median :15.50   Median :76.00   Median :1.000   toyota corolla    :  5
 Mean   :15.54   Mean   :75.98   Mean   :1.577   amc gremlin       :  4
 3rd Qu.:17.02   3rd Qu.:79.00   3rd Qu.:2.000   amc hornet        :  4
 Max.   :24.80   Max.   :82.00   Max.   :3.000   chevrolet chevette:  4
                                                 (Other)           :365
```

[1] James *et al.* (2021) use the `cv.glm()` function in the **boot** package (Canty and Ripley 2022; Davison and Hinkley 1997). Despite its name, `cv.glm()` is an independent function and not a method of a `cv()` generic function. The **boot** package is part of the standard R distribution.

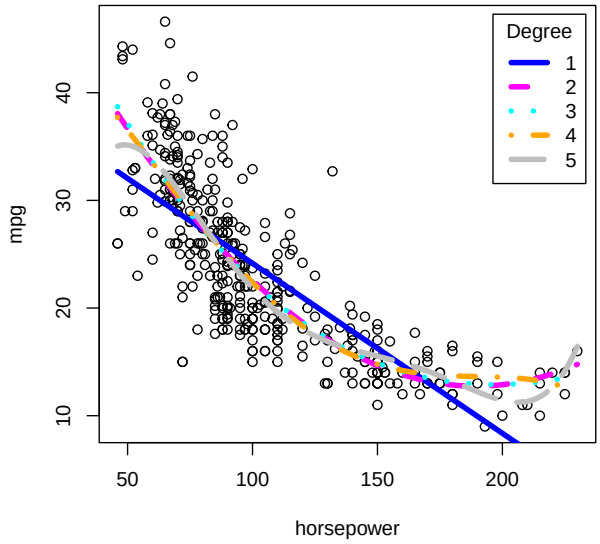


Figure 1: `mpg` vs. `horsepower` for the `Auto` data, showing fitted polynomials of degree 1 through 5.

With the exception of `origin` (which we don't use here), these variables are largely self-explanatory, except possibly for units of measurement: for details see `help("Auto", package="ISLR2")`.

We'll focus here on the relationship of `mpg` (miles per gallon) to `horsepower`, as displayed in Figure 1. The relationship between the two variables is monotone, decreasing, and nonlinear. Following James *et al.* (2021), we'll consider approximating the relationship by a polynomial regression, with the degree of the polynomial p ranging from 1 (a linear regression) to 10.[2] Polynomial fits for $p = 1$ to 5 are shown in Figure 1. The linear fit is clearly inappropriate; the fits for $p = 2$ (quadratic) through 4 are very similar; and the fit for $p = 5$ may over-fit the data by chasing one or two relatively high `mpg` values at the right (but see the CV results reported below).

Figure 2 shows two measures of estimated (squared) error as a function of polynomial-regression degree: The mean-squared error ("MSE"), defined as $\mathsf{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2$, and the usual residual variance, defined as $\widehat{\sigma}^2 = \frac{1}{n-p-1}\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2$. The former necessarily declines with p (or, more strictly, can't increase with p), while the latter gets slightly larger for the largest values of p, with the "best" value, by a small margin, for $p = 7$.

The generic `cv()` function has an `"lm"` method,

```
R> args(cv:::cv.lm)
```

```
function (model, data = insight::get_data(model), criterion = mse,
    k = 10L, reps = 1L, seed = NULL, details = k <= 10L, confint = n >=
        400L, level = 0.95, method = c("auto", "hatvalues", "Woodbury",
        "naive"), ncores = 1L, ...)
NULL
```

[2] Although it serves to illustrate the use of CV, a polynomial is not the best choice here. Consider, for example the scatterplot for log-transformed `mpg` and `horsepower`, produced by `plot(mpg ~ horsepower, data=Auto, log="xy")` (execution of which is left to the reader). We revisit the `Auto` data in Section 4.

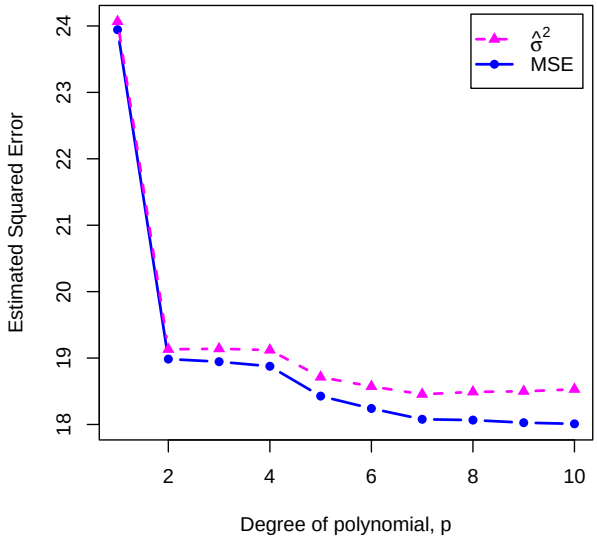


Figure 2: Estimated squared error as a function of polynomial degree, p

which takes the following arguments:

- `model`, an `"lm"` object, the only required argument.
- `data`, which can usually be inferred from the `model` object.
- `criterion`, a function to compute the CV criterion (defaulting to `mse`).
- `k`, the number of folds to employ (defaulting to `10`); the character value `"n"` or `"loo"` may be supplied to specify leave-one-out cross-validation.
- `reps`, the number of times to repeat the CV procedure (defaulting to 1).
- `seed`, the seed for R's pseudo-random number generator; if not specified a value is randomly selected, reported, and saved, so that the CV procedure is replicable.
- `confint`, whether or not to compute a confidence interval for the CV criterion, defaulting to `TRUE` if there are at least 400 cases; a confidence interval is computed only if the CV criterion can be expressed as the average of casewise components (see Section 7.2 for details).
- `level`, the level for the confidence interval (defaulting to `0.95`).
- `method`, the computational method to employ: `"hatvalues"` is relevant only for LOO CV and bases computation on the hatvalues for the linear model; `"Woodbury"` employs the Woodbury matrix identity to compute the CV criterion with each fold deleted; `"naive"` updates the model using the `update()` function; and `"auto"` selects `"hatvalues"` for LOO CV and `"Woodbury"` for k-fold CV, both of which are much more efficient than literally updating the least-squares fit (see below and Section 7.1).
- `ncores`, the number of cores to employ for parallel computation; if `cores = 1` (the default), the computations are not parallelized.

To illustrate, we perform 10-fold CV for a quadratic polynomial fit to the `Auto` data:

```
R> m.auto <- lm(mpg ~ poly(horsepower, 2), data=Auto)
R> summary(m.auto)


Call:
lm(formula = mpg ~ poly(horsepower, 2), data = Auto)

Residuals:
     Min       1Q   Median       3Q      Max
-14.7135  -2.5943  -0.0859   2.2868  15.8961

Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)            23.4459     0.2209  106.13   <2e-16
poly(horsepower, 2)1 -120.1377     4.3739  -27.47   <2e-16
poly(horsepower, 2)2   44.0895     4.3739   10.08   <2e-16

Residual standard error: 4.374 on 389 degrees of freedom
Multiple R-squared:  0.6876,    Adjusted R-squared:  0.686
F-statistic:   428 on 2 and 389 DF,  p-value: < 2.2e-16

R> cv(m.auto, confint=TRUE)

R RNG seed set to 968919

10-Fold Cross Validation
method: Woodbury
criterion: mse
cross-validation criterion = 19.22048
bias-adjusted cross-validation criterion = 19.20805
95% CI for bias-adjusted CV criterion = (15.74739, 22.66871)
full-sample criterion = 18.98477
```

The function reports the CV estimate of MSE, a bias-adjusted estimate of the MSE (the bias adjustment is explained in Section 7.2), and the MSE is also computed for the original, full-sample regression. Because the number of cases $n = 392 < 400$ for the `Auto` data, we set the argument `confint=TRUE` to obtain a confidence interval for the MSE, which proves to be quite wide.

To perform LOO CV:

```
R> cv(m.auto, k="loo")

n-Fold Cross Validation
method: hatvalues
criterion: mse
cross-validation criterion = 19.24821
```

The "hatvalues" method reports only the CV estimate of MSE. Alternative methods are to use the Woodbury matrix identity or the "naive" approach of literally refitting the model with each case omitted. All three methods produce exact results for a linear model (within the precision of floating-point computations):

```
R> cv(m.auto, k="loo", method="naive", confint=TRUE)

n-Fold Cross Validation
method: naive
criterion: mse
cross-validation criterion = 19.24821
bias-adjusted cross-validation criterion = 19.24787
95% CI for bias-adjusted CV criterion = (15.77884, 22.71691)
full-sample criterion = 18.98477

R> cv(m.auto, k="loo", method="Woodbury", confint=TRUE)

n-Fold Cross Validation
method: Woodbury
criterion: mse
cross-validation criterion = 19.24821
bias-adjusted cross-validation criterion = 19.24787
95% CI for bias-adjusted CV criterion = (15.77884, 22.71691)
full-sample criterion = 18.98477
```

The "naive" and "Woodbury" methods also return the bias-adjusted estimate of MSE (and a confidence interval around it) along with the full-sample MSE, but bias isn't an issue for LOO CV.

This is a small regression problem and all three computational approaches are essentially instantaneous, but it is still of interest to investigate their relative speed. In the following comparison, we include the cv.glm() function from the **boot** package, which takes the naive approach, and for which we have to fit the linear model as an equivalent Gaussian GLM. We use the microbenchmark() function from the package of the same name (Mersmann 2023) for the timings:

```
R> m.auto.glm <- glm(mpg ~ poly(horsepower, 2), data=Auto)
R> boot::cv.glm(Auto, m.auto.glm)$delta # MSE, biased-corrected MSE

[1] 19.24821 19.24787

R> microbenchmark::microbenchmark(
+   hatvalues = cv(m.auto, k="loo"),
+   Woodbury = cv(m.auto, k="loo", method="Woodbury"),
+   naive = cv(m.auto, k="loo", method="naive"),
+   cv.glm = boot::cv.glm(Auto, m.auto.glm),
+   times=10
+ )
```

```
Warning in microbenchmark::microbenchmark(hatvalues = cv(m.auto, k = "loo"), :
less accurate nanosecond times to avoid potential integer overflows

Unit: microseconds
      expr        min         lq       mean     median         uq        max
 hatvalues    992.282   1149.681   1181.427   1199.188   1223.235   1356.485
  Woodbury  12297.663  12375.317  14678.447  12908.952  16017.675  22227.535
     naive 217204.347 222086.545 249652.690 250411.416 277112.440 279141.530
    cv.glm 379237.823 382052.063 401064.952 388823.726 435052.394 440563.573
 neval
    10
    10
    10
    10
```

On our computer, using the hatvalues is about an order of magnitude faster than employing Woodbury matrix updates, and more than two orders of magnitude faster than refitting the model.

2.1. Comparing competing models

The `cv()` function also has a method that can be applied to a list of regression models for the same data, composed using the `models()` function. For k-fold CV, the same folds are used for the competing models, which reduces random error in their comparison. This result can also be obtained by specifying a common seed for R's random-number generator while applying `cv()` separately to each model, but employing a list of models is more convenient for both k-fold and LOO CV (where there is no random component to the composition of the n folds).

We illustrate with the polynomial regression models of varying degree for the `Auto` data, beginning by fitting and saving the 10 models:

```
R> mlist <- vector(10, mode="list")
R> for (p in 1:10) mlist[[p]] <- lm(mpg ~ poly(horsepower, p), data = Auto)
R> names(mlist) <- paste0("m.", 1:10)
R> mlist[2] # e.g., the quadratic fit

$m.2

Call:
lm(formula = mpg ~ poly(horsepower, p), data = Auto)

Coefficients:
         (Intercept)  poly(horsepower, p)1  poly(horsepower, p)2
               23.45                -120.14                 44.09
```

We then apply `cv()` to the list of 10 models (the `data` argument is required):

```
R> # 10-fold CV
R> mlist <- do.call(models, mlist) # create "modList" object
R>
R> cv.auto.10 <- cv(mlist, data=Auto, seed=2120)
R> cv.auto.10[2] # e.g., for quadratic model


Model m.2:
10-Fold Cross Validation
method: Woodbury
cross-validation criterion = 19.34601
bias-adjusted cross-validation criterion = 19.32699
full-sample criterion = 18.98477

R> # LOO CV
R> cv.auto.loo <- cv(mlist, data=Auto, k="loo")
R> cv.auto.loo[2] # e.g., for quadratic model


Model m.2:
n-Fold Cross Validation
method: hatvalues
cross-validation criterion = 19.24821
```

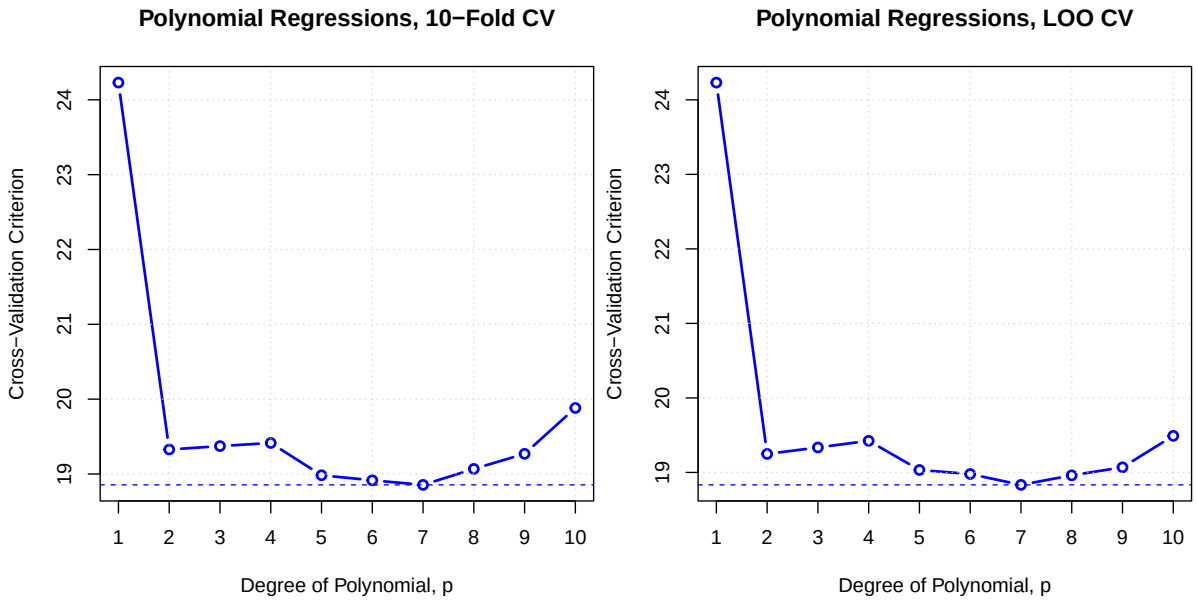
The `models()` function takes an arbitrary number of regression models as its arguments, which are optionally named, to create a `"modList"` object. Because we generated the polynomial regression models in a named list, we conveniently employ `do.call()` to supply the models as arguments to `models()`. The names created for the list (e.g., `"m.2"`) are then used for the models. We can also invoke the `plot()` method for `"cvModList"` objects to compare the models (see Figure 3):

```
R> plot(cv.auto.10, main="Polynomial Regressions, 10-Fold CV",
+      axis.args=list(labels=1:10), xlab="Degree of Polynomial, p")
R> plot(cv.auto.loo, main="Polynomial Regressions, LOO CV",
+      axis.args=list(labels=1:10), xlab="Degree of Polynomial, p")
```

In this example, 10-fold and LOO CV produce generally similar results, and also results that are similar to those produced by the estimated error variance $\widehat{\sigma}^2$ for each model (cf., Figure 2 on page 5), except for the highest-degree polynomials, where the CV results more clearly suggest over-fitting.

3. Cross-validating mixed-effects models

The fundamental analogy for cross-validation is to the collection of new data. That is, predicting the response in each fold from the model fit to data in the other folds is like using the model fit to all of the data to predict the response for new cases from the values of the



(a) (b)

Figure 3: Cross-validated (a) 10-fold and (b) LOO MSE as a function of polynomial degree, p.

predictors for those new cases. As we explained, the application of this idea to independently sampled cases is straightforward.

In contrast, mixed-effects models are fit to *dependent* data, in which cases are clustered, such as hierarchical data, where the clusters comprise higher-level units (e.g., students clustered in schools), or longitudinal data, where the clusters are individuals and the cases are repeated observations on the individuals over time.[3]

We can think of two approaches to applying cross-validation to clustered data:[4]

1. Treat CV as analogous to predicting the response for one or more cases in a *newly observed cluster*. In this instance, the folds comprise one or more whole clusters; we refit the model with all of the cases in clusters in the current fold removed; and then we predict the response for the cases in clusters in the current fold. These predictions are based only on fixed effects because the random effects for the omitted clusters are construed to be unknown, as they would be for data on cases in newly observed clusters.

2. Treat CV as analogous to predicting the response for a newly observed case in an *existing cluster*. In this instance, the folds comprise one or more individual cases, and the

[3]There are, however, more complex situations that give rise to so-called *crossed* (rather than *nested*) random effects. For example, consider students within classes within schools. In primary schools, students typically are in a single class, and so classes are nested within schools. In secondary schools, however, students typically take several classes and students who are together in a particular class may not be together in other classes; consequently, random effects based on classes within schools are crossed. The `lmer()` function in the **lme4** package, for example, is capable of modeling both nested and crossed random effects, and the `cv()` methods for mixed models in the **cv** package pertain to both nested and crossed random effects. We present an example of the latter in a vignette for the **cv** package.

[4]We subsequently discovered that Vehtari (2023, Section 8) makes similar points.

predictions can use both the fixed and random effects—so-called "best-linear-unbiased predictors" or "BLUPs."

3.1. Example: The High-School and Beyond data

Following their use by Raudenbush and Bryk (2002), data from the 1982 *High School and Beyond* (HSB) survey have become a staple of the literature on mixed-effects models. The HSB data are used by Fox and Weisberg (2019, Sec. 7.2.2) to illustrate the application of linear mixed models to hierarchical data, and we'll closely follow their example here.

The HSB data are included in the `MathAchieve` and `MathAchSchool` data sets in the **nlme** package (Pinheiro and Bates 2000). `MathAchieve` comprises individual-level data on 7185 students in 160 high schools, and `MathAchSchool` contains school-level data:

```
R> data("MathAchieve", package="nlme")
R> dim(MathAchieve)

[1] 7185    6

R> head(MathAchieve, 3)

Grouped Data: MathAch ~ SES | School
  School Minority    Sex    SES MathAch MEANSES
1   1224       No Female -1.528   5.876  -0.428
2   1224       No Female -0.588  19.708  -0.428
3   1224       No   Male -0.528  20.349  -0.428

R> tail(MathAchieve, 3)

Grouped Data: MathAch ~ SES | School
     School Minority    Sex    SES MathAch MEANSES
7183   9586       No Female  1.332  19.641   0.627
7184   9586       No Female -0.008  16.241   0.627
7185   9586       No Female  0.792  22.733   0.627

R> data("MathAchSchool", package="nlme")
R> dim(MathAchSchool)

[1] 160   7

R> head(MathAchSchool, 2)

     School Size Sector PRACAD DISCLIM HIMINTY MEANSES
1224   1224  842 Public   0.35   1.597       0  -0.428
1288   1288 1855 Public   0.27   0.174       0   0.128

R> tail(MathAchSchool, 2)
```

```
     School Size   Sector PRACAD DISCLIM HIMINTY MEANSES
9550   9550 1532   Public   0.45   0.791       0   0.059
9586   9586  262 Catholic   1.00  -2.416       0   0.627
```

The first few students are in school number 1224 and the last few in school 9586.

We'll use only the `School`, `SES` (students' socioeconomic status), and `MathAch` (their score on a standardized math-achievement test) variables in the `MathAchieve` data set, and `Sector` (`"Catholic"` or `"Public"`) in the `MathAchSchool` data set.

Some data-management is required before fitting a mixed-effects model to the HSB data:

```
R> HSB <- MathAchieve
R> HSB <- merge(MathAchSchool[, c("School", "Sector")],
+               HSB[, c("School", "SES", "MathAch")], by="School")
R> names(HSB) <- tolower(names(HSB))
R> HSB <- within(HSB, {
+   mean.ses <- ave(ses, school)
+   cses <- ses - mean.ses
+ })
```

In the process, we merge variables from the school-level and student-level data sets, and create two new school-level variables: `mean.ses`, which is the average SES for students in each school; and `cses`, which is the individual students' SES centered at their school means. For details, see Fox and Weisberg (2019, Sec. 7.2.2).

Still following Fox and Weisberg, we proceed to use the `lmer()` function in the **lme4** package (Bates, Mächler, Bolker, and Walker 2015) to fit a mixed model for math achievement to the HSB data:

```
R> library("lme4", quietly=TRUE)
R> hsb.lmer <- lmer(mathach ~ mean.ses*cses + sector*cses
+                     + (cses | school), data=HSB)
R> summary(hsb.lmer, correlation=FALSE)

Linear mixed model fit by REML ['lmerMod']
Formula: mathach ~ mean.ses * cses + sector * cses + (cses | school)
   Data: HSB

REML criterion at convergence: 46503.7

Scaled residuals:
     Min       1Q   Median       3Q      Max
-3.15926 -0.72319  0.01704  0.75444  2.95822

Random effects:
 Groups   Name        Variance Std.Dev. Corr
 school   (Intercept)  2.380   1.5426
          cses         0.101   0.3179   0.39
```

```
 Residual             36.721   6.0598
Number of obs: 7185, groups:  school, 160

Fixed effects:
                    Estimate Std. Error t value
(Intercept)          12.1279     0.1993  60.856
mean.ses              5.3329     0.3692  14.446
cses                  2.9450     0.1556  18.928
sectorCatholic        1.2266     0.3063   4.005
mean.ses:cses         1.0393     0.2989   3.477
cses:sectorCatholic  -1.6427     0.2398  -6.851
```

We can then cross-validate at the cluster (i.e., school) level,

```
R> cv(hsb.lmer, k=10, clusterVariables="school", seed=5240)

R RNG seed set to 5240

10-Fold Cross Validation based on 160 {school} clusters
criterion: mse
cross-validation criterion = 39.15662
bias-adjusted cross-validation criterion = 39.14844
95% CI for bias-adjusted CV criterion = (38.06554, 40.23135)
full-sample criterion = 39.00599
```

or at the case (i.e., student) level,

```
R> cv(hsb.lmer, seed=1575)

R RNG seed set to 1575

Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
Model failed to converge with max|grad| = 0.00587207 (tol = 0.002, component 1)

boundary (singular) fit: see help('isSingular')

10-Fold Cross Validation
criterion: mse
cross-validation criterion = 37.44473
bias-adjusted cross-validation criterion = 37.33801
95% CI for bias-adjusted CV criterion = (36.28761, 38.38841)
full-sample criterion = 36.06767
```

For cluster-level CV, the `clusterVariables` argument tells `cv()` how the clusters are defined. Were there more than one clustering variable, say classes within schools, these would be provided as a character vector of variable names: `clusterVariables = c("school",`

`"class"`). For cluster-level CV, the default is `k = "loo"`, that is, leave one cluster out at a time; we instead specify `k = 10` folds of clusters, each fold therefore comprising $160/10 = 16$ schools.

If the `clusterVariables` argument is omitted, then case-level CV is employed, with `k = 10` folds as the default, here each with $7185/10 \approx 719$ students. Notice that one of the 10 models refit with a fold removed failed to converge. Convergence problems are common in mixed-effects modeling. The issue here is that an estimated variance component is close to or equal to 0, which is at a boundary of the parameter space. That shouldn't disqualify the fitted model for the kind of prediction required for cross-validation.

`cv()` also has methods for mixed models fit by the `glmer()` function in the **lme4** package, the `lme()` function in the **nlme** package (Pinheiro and Bates 2000), and the `glmmTMB()` function in the **glmmTMB** package (Brooks, Kristensen, van Benthem, Magnusson, Berg, Nielsen, Skaug, Maechler, and Bolker 2017), along with a simple procedure for extending `cv()` to other classes of mixed-effects models. See the vignettes in the **cv** package for details.

3.2. Example: Contrasting cluster-based and case-based CV

In this section, we introduce an artificial data set that exemplifies aspects of cross-validation particular to hierarchical models. Using this data set, we show that model comparisons employing cluster-based and those employing case-based cross-validation may not agree on a "best" model. Furthermore, commonly used measures of fit, such as mean-squared error, do not necessarily become smaller as models become larger, even when the models are nested, and even when the measure of fit is computed for the whole data set.

Consider a researcher studying improvement in a skill, singing, for example, among students enrolled in a four-year voice program at a music conservatory. The plan is to measure each student's skill level at the beginning of the program and every year thereafter until the end of the program, resulting in 5 annual measurements for each student. It turns out that singing appeals to students of all ages, and students enrolling in the program range in age from 20 to 70. Moreover, participants' untrained singing skill is similar at all ages, as is their rate of progress with training. All students complete the four-year program.

The researcher, who has more expertise in singing than in modeling, decides to model the response, y, singing skill, as a function of age, x, reasoning that students get older during their stay in the program, and (incorrectly) that age can serve as a proxy for elapsed time. The researcher knows that a mixed model should be used to account for clustering due to the expected similarity of measurements taken from each student.

We start by generating the data, using parameters consistent with the description above and meant to highlight the issues that arise in cross-validating mixed-effects models:[5]

```
R> # Parameters:
R> set.seed(9693)
R> Nb <- 100      # number of groups
R> Nw <- 5        # number of individuals within groups
R> Bb <- 0        # between-group regression coefficient on group mean
R> SDre <- 2.0    # between-group SD of random level relative to group mean of x
```

[5] We invite the interested reader to experiment with varying the parameters of our example.

```
R> SDwithin <- 0.5  # within group SD
R> Bw <- 1          # within group effect of x
R> Ay <- 10         # intercept for response
R> Ax <- 20         # starting level of x
R> Nx <- Nw*10      # number of distinct x values
R>
R> Data <- data.frame(
+   group = factor(rep(1:Nb, each=Nw)),
+   x = Ax + rep(1:Nx, length.out = Nw*Nb)
+ ) |> within ({
+   xm  <- ave(x, group, FUN = mean) # within-group mean
+   y <- Ay +
+     Bb * xm +                    # contextual effect
+     Bw * (x - xm) +              # within-group effect
+     rnorm(Nb, sd=SDre)[group] +  # random level by group
+     rnorm(Nb*Nw, sd=SDwithin)    # random error within groups
+ })
```

Figure 4 (a) shows a scatterplot the data for a representative group of 10 (without loss of generality, the first 10) of the 100 students, displaying the 95% concentration ellipse for each cluster.

The between-student effect of age is 0 but the within-student effect is 1. Due to the large variation in ages between students, the least-squares regression of singing skill on age (for the 500 observations among all 100 students) produces an estimated slope close to 0 (though with a small p-value), because the slope is heavily weighted toward the between-student effect:

```
R> summary(lm(y ~ x, data=Data))


Call:
lm(formula = y ~ x, data = Data)

Residuals:
    Min      1Q  Median      3Q     Max
-5.7713 -1.6583 -0.0894  1.5520  7.6240

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 9.050430   0.347189  26.068  < 2e-16
x           0.020908   0.007273   2.875  0.00422

Residual standard error: 2.347 on 498 degrees of freedom
Multiple R-squared:  0.01632,   Adjusted R-squared:  0.01435
F-statistic: 8.263 on 1 and 498 DF,  p-value: 0.004219
```

We proceed to fit several mixed-effects models to the data, using the `compareCoefs()` function from the **car** package (Fox and Weisberg 2019) to display the fixed-effect estimates for these models. We discuss each of these models below.

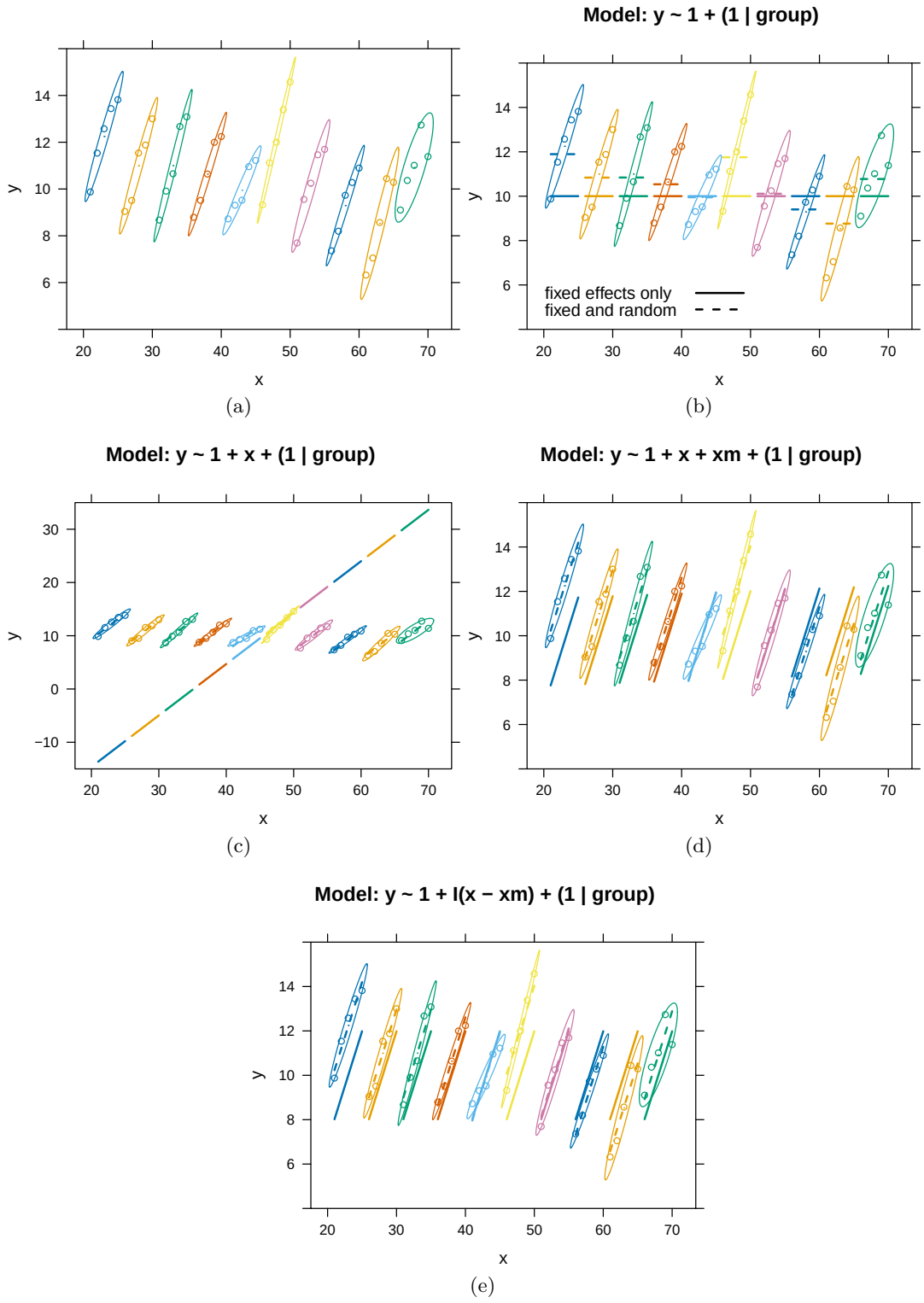


Figure 4: (a) Hierarchical data set, showing the first 10 of 100 students, and (b)–(e) predictions from several mixed models fit to the data

```
R> mod.0 <- lmer(y ~ 1 + (1 | group), Data)
R> mod.1 <- lmer(y ~ x + (1 | group), Data)
R> mod.2 <- lmer(y ~ x + xm + (1 | group), Data)
R> mod.3 <- lmer(y ~ I(x - xm) + (1 | group), Data)
R> library("car", quietly=TRUE)
R> compareCoefs(mod.0, mod.1, mod.2, mod.3)

Calls:
1: lmer(formula = y ~ 1 + (1 | group), data = Data)
2: lmer(formula = y ~ x + (1 | group), data = Data)
3: lmer(formula = y ~ x + xm + (1 | group), data = Data)
4: lmer(formula = y ~ I(x - xm) + (1 | group), data = Data)

            Model 1 Model 2 Model 3 Model 4
(Intercept)  10.002 -33.919   9.479  10.002
SE            0.186   1.564   0.617   0.186

x                    0.9653  0.9915
SE                   0.0158  0.0160

xm                          -0.9800
SE                           0.0206

I(x - xm)                                0.992
SE                                       0.016
```

The initial mixed-effects model, `mod.0`, is a simple random-intercepts model. We obtain predictions from this model for the fixed effects alone, as would be used for cross-validation based on clusters (i.e., students), and for fixed and random effects—the BLUPs—as would be used for cross-validation based on cases (i.e., occasions within students). Predictions from `mod.0` for the first 10 students are shown in Figure 4 (b). The fixed-effect predictions for the various individuals are identical—the estimated fixed-effects intercept or estimated general mean of y—while the BLUPs are the sums of the fixed-effects intercept and the random intercepts, and are only slightly shrunken towards the general mean. Because in our artificial data there is no population relationship between age and skill, the fixed-effect-only predictions and the BLUPs are not very different.

Our next model, `mod.1`, includes a fixed intercept and the fixed effect of `x`, along with a random intercept. Predictions from this model appear in Figure 4 (c). The BLUPs fit the observed data very closely, but predictions based on the fixed effects alone, with a common intercept and slope for all clusters, are very poor—indeed, much worse than the fixed-effects-only predictions based on the simpler random-intercept model, `mod.0`. We therefore anticipate (and show later in this section) that case-based cross-validation will prefer `mod.1` to `mod.0`, but that cluster-based cross-validation will prefer `mod.0` to `mod.1`.

Our third model, `mod.2`, includes the "contextual effect" of x—that is, the cluster mean `xm`—along with x and the intercept in the fixed-effect part of the model, and a random intercept.

This model is equivalent to fitting `y ~ I(x - xm) + xm + (1 | group)`, which is the model that generated the data once the coefficient of the contextual predictor `xm` is set to 0 (as it is in `mod.3`, discussed below).

Predictions from `mod.2` appear in Figure 4 (d). Depending on the estimated variance parameters of the model, a mixed model like `mod.2` will apply varying degrees of shrinkage to the random-intercept BLUPs that correspond to variation in the heights of the parallel fitted lines for the individual students. In our contrived data, `mod.2` applies little shrinkage, allowing substantial variability in the heights of the fitted lines, which closely approach the observed values for each student. The fit of the mixed model `mod.2` is consequently similar to that of a fixed-effects model with age and a categorical predictor for individual students (i.e., treating students as a factor, and not shown here).

The mixed model `mod.2` therefore fits the individual observations well, and we anticipate a favorable assessment using individual-based cross-validation. In contrast, the large variability in the BLUPs results in larger residuals for predictions based on fixed effects alone, and so we expect that cluster-based cross-validation won't show an advantage for `mod.2` compared to the smaller `mod.0`, which includes only fixed and random intercepts.

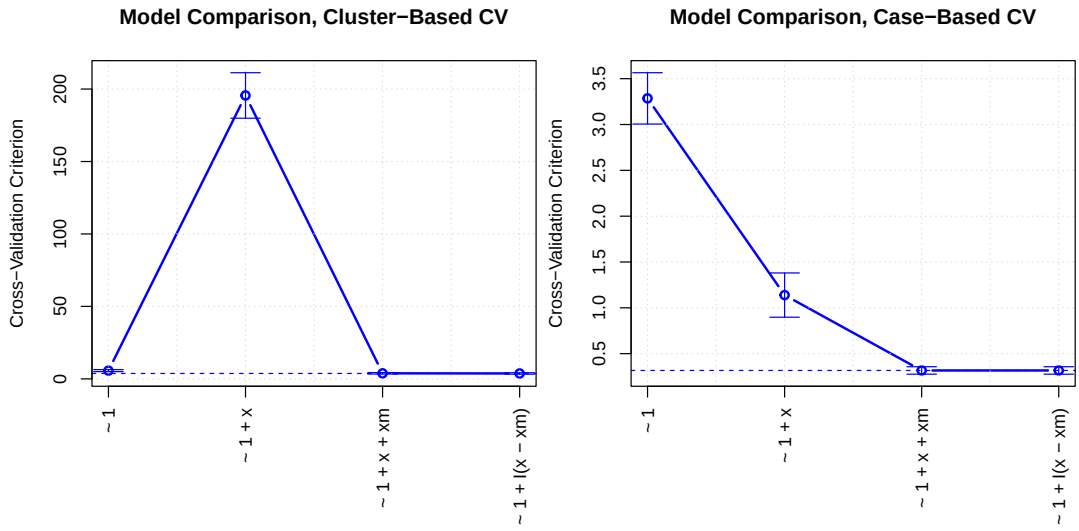
Had the mixed model applied considerable shrinkage, then neither cluster-based nor case-based cross-validation would show much improvement over the random-intercept-only model. In our experience, the degree of shrinkage does not vary smoothly as parameters are changed but tends to be "all or nothing," and near the tipping point, the behavior of estimates can be affected considerably by the choice of algorithm used to fit the model.

Finally, `mod.3` directly estimates the model used to generate the data. As mentioned, it is a constrained version of `mod.2`, with the coefficient of `xm` set to 0, and with `x` expressed as a deviation from the cluster mean `xm`. The predictions from `mod.3`, shown in Figure 4 (e), are therefore similar to those from `mod.2`.

We next carry out case-based cross-validation, which, as we have explained, includes both fixed and predicted random effects (i.e., BLUPs), and cluster-based cross-validation, which includes fixed effects only. In order to reduce between-model random variability in comparisons of models, we apply `cv()` to the list of models created by the `models()` function (introduced previously), performing cross-validation with the same folds for each model (see Figure 5):

```
R> modlist <- models("~ 1"=mod.0, "~ 1 + x"=mod.1,
+                    "~ 1 + x + xm"=mod.2, "~ 1 + I(x - xm)"=mod.3)
R>
R> cvs_clusters <- cv(modlist, data=Data, cluster="group", k=10, seed=6449)
R> plot(cvs_clusters, main="Model Comparison, Cluster-Based CV")
R>
R> cvs_cases <- cv(modlist, data=Data, seed=9693)
R> plot(cvs_cases, main="Model Comparison, Case-Based CV")
```

In summary, `mod.1`, with x alone and without the contextual mean of x, is assessed as fitting very poorly by cluster-based CV, but relatively much better by case-based CV. `mod.2`, which includes both x and its contextual mean, produces better results using both cluster-based and case-based CV. The data-generating model, `mod.3`, which includes the fixed effect of `x - xm` in place of separate terms in `x` and `xm`, isn't distinguishable from `mod.2`, which includes `x` and `xm` separately, even though `mod.2` has an unnecessary parameter (recall that



(a) (b)

Figure 5: 10-fold (a) cluster-based and (b) case-based cross-validation comparing random intercept models with varying fixed effects. The error bars show the 95% confidence interval around the CV estimate of the MSE for each model.

the population coefficient of `xm` is 0 when `x` is expressed as deviations from the contextual mean). These conclusions are consistent with our observations based on graphing predictions from the various models in Figure 4 (on page 16), and they illustrate the desirability of assessing mixed-effect models at different hierarchical levels.

4. Cross-validating model specification

As Hastie, Tibshirani, and Friedman (2009, Sec. 7.10.2: "The Wrong and Right Way to Do Cross-validation") explain, if the whole data are used to specify or fine-tune a statistical model, then subsequent cross-validation of the model is intrinsically misleading, because the model is selected to fit the whole data, including the part of the data that remains when each fold is removed. Statistical modeling is partly a craft, and one could imagine applying that craft to successive partial data sets, each with a fold removed. The resulting procedure would be tedious, though possibly worth the effort, but it would also be difficult to realize in practice: After all, we can hardly erase our memory of statistical modeling choices between analyzing partial data sets. Alternatively, if we're able to automate the process of model specification, then we can more realistically apply CV mechanically.

The `"function"` method for `cv()` cross-validates a model-specification process in a general manner. Functions for four such model-specification processes are included in the package: `selectStepAIC()`, based on the `stepAIC()` function in the **MASS** package (Venables and Ripley 2002), performs stepwise predictor selection for regression models; `selectTrans()`, based on the `powerTransform()` function in the **car** package (Fox and Weisberg 2019), transforms predictors and the response in a regression model towards normality; `selectTransAndStepAIC()`—the use of which we illustrate in the current section—performs both of these procedures se-

quentially; and `selectModelList()`—also illustrated in the current section—uses CV both to select one of several competing models, and then, recursively, to estimate prediction error for the selected model. In a vignette on extending the **cv** package, we explain how to specify additional model-selection procedures.

4.1. Example: Data transformation and predictor selection for the Auto data

To illustrate cross-validation of model specification, we return to the `Auto` data set:[6]

```
R> names(Auto)

[1] "mpg"          "cylinders"    "displacement" "horsepower"   "weight"
[6] "acceleration" "year"         "origin"       "name"

R> xtabs(~ year, data=Auto)

year
70 71 72 73 74 75 76 77 78 79 80 81 82
29 27 28 40 26 30 34 28 36 29 27 28 30

R> xtabs(~ origin, data=Auto)

origin
  1   2   3
245  68  79

R> xtabs(~ cylinders, data=Auto)

cylinders
  3   4   5   6   8
  4 199   3  83 103
```

The `Auto` data appeared in a preliminary example in Section 2, where we employed CV to inform the selection of the degree of a polynomial regression of `mpg` on `horsepower`. Here, we consider more generally the problem of predicting `mpg` from the other variables in the `Auto` data. We begin with a bit of data management, and then examine the pairwise relationships among the numeric variables in the data set (Figure 6, produced by the `scatterplotMatrix()` function in the **car** package):

```
R> Auto$cylinders <- factor(Auto$cylinders,
+                           labels=c("3-4", "3-4", "5-6", "5-6", "8"))
R> Auto$year <- as.factor(Auto$year)
R> Auto$origin <- factor(Auto$origin,
```

[6]This example benefits from an email conversation with Bill Venables, who of course isn't responsible for the use to which we've put his insightful remarks.

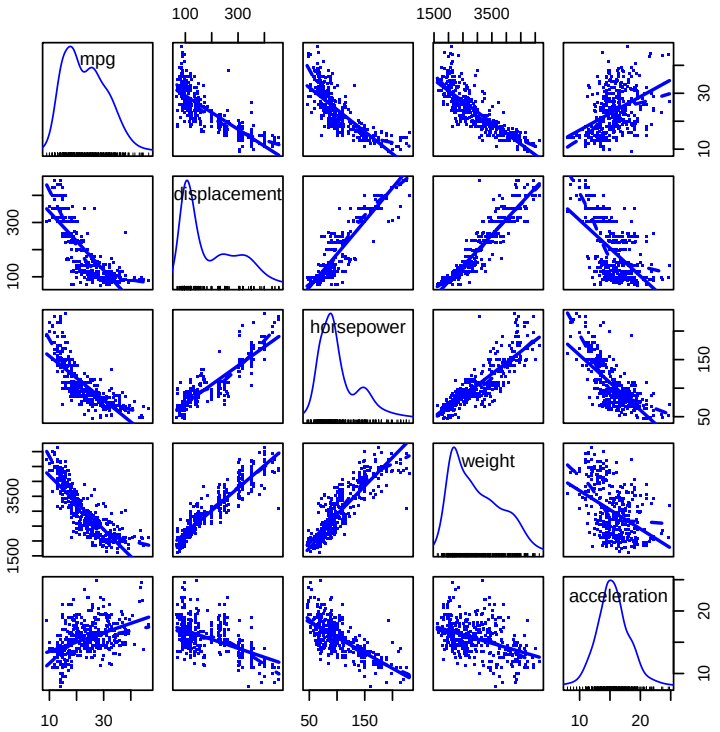


Figure 6: Scatterplot matrix for the numeric variables in the `Auto` data. In each panel, the solid line shows the linear least-squares fit and the broken line is for a nonparametric regression.

```
+                       labels=c("America", "Europe", "Japan"))
R> rownames(Auto) <- make.names(Auto$name, unique=TRUE)
R> Auto$name <- NULL
R>
R> scatterplotMatrix(~ mpg + displacement + horsepower + weight + acceleration,
+                    smooth=list(spread=FALSE), data=Auto, pch=".")
```

A comment before we proceed: `origin` is clearly categorical and so converting it to a factor is natural, but we could imagine treating `cylinders` and `year` as numeric predictors. There are, however, only 5 distinct values of `cylinders` (ranging from 3 to 8), but cars with 3 or 5 cylinders are rare, and none of the cars has 7 cylinders. There are similarly only 13 distinct years between 1970 and 1982 in the data, and the relationship between `mpg` and `year` is difficult to characterize.[7] It's apparent that most these variables are positively skewed and that many of the pairwise relationships among them are nonlinear.

We start with a "working model" that specifies linear partial relationships of the response to the numeric predictors:

[7]Making the decision to treat `year` as a factor on this basis could be construed as cheating in the current context, which illustrates the difficulty of automating the whole model-selection process. It's rarely desirable, in our opinion, to forgo exploration of the data to ensure the purity of model validation. We believe, however, that it's still useful to automate as much of the process as we can, as illustrated here, to obtain a more realistic, if still biased, estimate of the predictive power of a model.

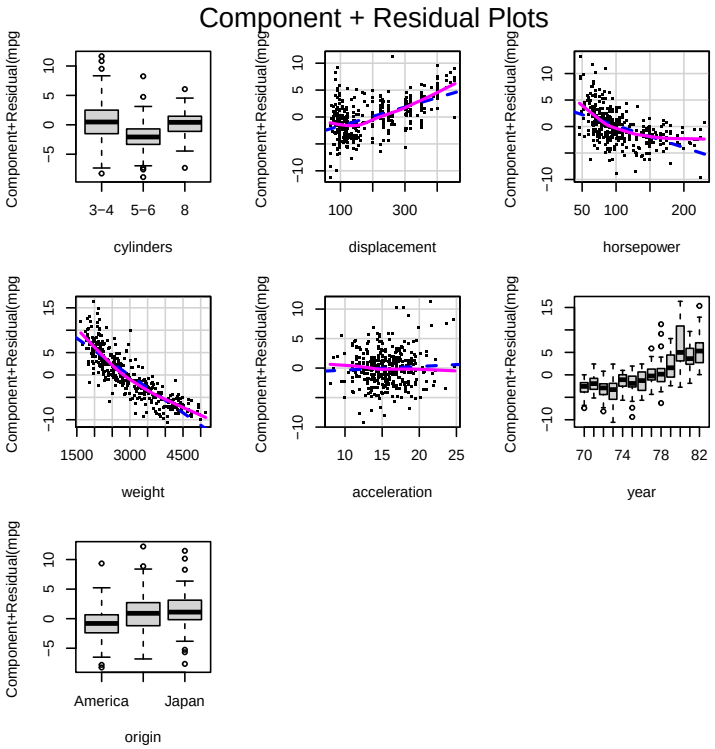


Figure 7: Component+residual plots for the working model fit to the `Auto` data. The broken blue lines show linear least-squares fits, while the solid magenta lines are nonparametric-regression smooths.

```
R> m.auto <- lm(mpg ~ ., data = Auto)
R> crPlots(m.auto, pch=".")
```

The component+residual plots in Figure 7 clearly reveal the inadequacy of the model.

Some background: As Weisberg (2014, Sec. 8.2) explains, there are technical advantages to having (numeric) predictors in linear regression analysis that are themselves linearly related. If the predictors *aren't* linearly related, then the relationships between them can often be straightened by power transformations. Transformations can be selected after graphical examination of the data, or by analytic methods, such as transforming the predictors towards multivariate normality, which implies linearity. Once the relationships between the predictors are linearized, it can be advantageous similarly to transform the conditional distribution of the response variable towards normality. Selecting transformations analytically raises the possibility of automating the process, as required for cross-validation.

The `powerTransform()` function in the **car** package transforms variables towards multivariate normality by a generalization of Box and Cox's maximum-likelihood-like approach (Box and Cox 1964). Several "families" of power transformations can be used, including the original Box-Cox family (which is the default), simple powers (and roots), and two adaptations of the Box-Cox family to data that may include negative values and 0s: the Box-Cox-with-negatives family and the Yeo-Johnson family; see Weisberg (2014, Chap. 8) and Fox and Weisberg (2019, Chap. 3) for details. We proceed to transform the numeric predictors in the `Auto` regression towards multivariate normality:

```
R> num.predictors <- c("displacement", "horsepower", "weight",
+                      "acceleration")
R> tr.x <- powerTransform(Auto[, num.predictors])
R> summary(tr.x)

bcPower Transformations to Multinormality
             Est Power Rounded Pwr Wald Lwr Bnd Wald Upr Bnd
displacement   -0.0509           0      -0.2082       0.1065
horsepower     -0.1249           0      -0.2693       0.0194
weight         -0.0870           0      -0.2948       0.1208
acceleration    0.3061           0      -0.0255       0.6376

Likelihood ratio test that transformation parameters are equal to 0
 (all log transformations)
                                LRT df    pval
LR test, lambda = (0 0 0 0) 4.872911  4 0.30059

Likelihood ratio test that no transformations are needed
                                LRT df       pval
LR test, lambda = (1 1 1 1) 390.0777  4 < 2.22e-16
```

We then apply the (rounded) transformations—all, as it turns out, logs (i.e., "Oth" powers)—to the data and re-estimate the model:

```
R> A <- Auto
R> powers <- tr.x$roundlam
R> for (pred in num.predictors){
+   A[, pred] <- bcPower(A[, pred], lambda=powers[pred])
+ }
R> m <- update(m.auto, data=A)
```

Having transformed the predictors towards multivariate normality, we now consider whether there's evidence for transforming the response (using `powerTransform()` for Box and Cox's original method), also obtaining a log transformation:

```
R> summary(powerTransform(m))

bcPower Transformation to Normality
   Est Power Rounded Pwr Wald Lwr Bnd Wald Upr Bnd
Y1    0.0024           0      -0.1607       0.1654

Likelihood ratio test that transformation parameter is equal to 0
 (log transformation)
                               LRT df    pval
LR test, lambda = (0) 0.0008015428  1 0.97741
```

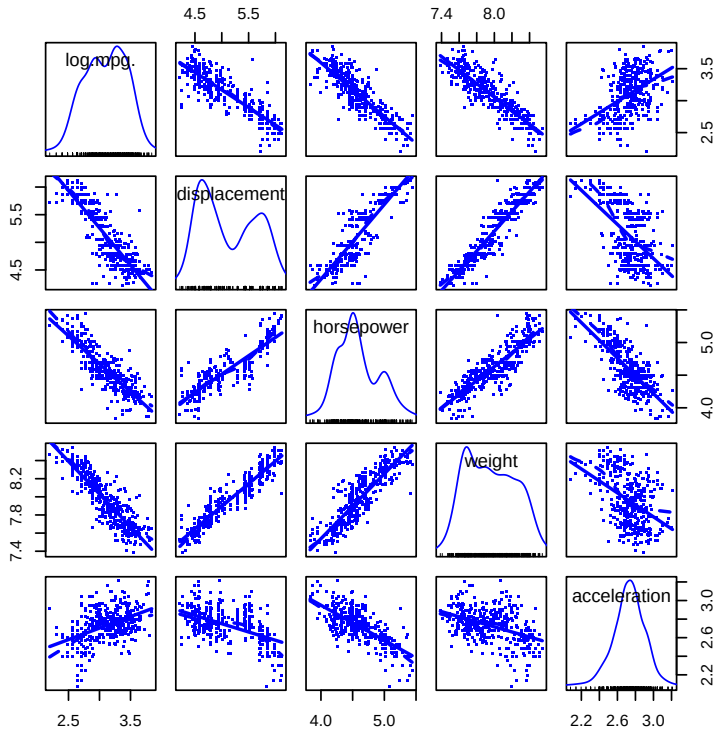


Figure 8: Scatterplot matrix for the transformed numeric variables in the `Auto` data

```
Likelihood ratio test that no transformation is needed
                           LRT df       pval
LR test, lambda = (1) 124.1307  1 < 2.22e-16
```

```
R> m <- update(m, log(mpg) ~ .)
```

The transformed numeric variables are much better-behaved (cf., Figure 6, on page 21, and Figure 8):

```
R> scatterplotMatrix(~ log(mpg) + displacement + horsepower + weight
+                    + acceleration,
+                    smooth=list(spread=FALSE), data=A, pch=".")
```

And the partial relationships in the model fit to the transformed data are much more nearly linear (cf., Figure 7, on page 22, and Figure 9):

```
R> crPlots(m, pch=".")
```

After transforming both the numeric predictors and the response, we proceed to use the `stepAIC()` function in the **MASS** package to perform predictor selection, employing the BIC model-selection criterion (by setting the `k` argument of `stepAIC()` to $\log n$):

```
R> library("MASS")
R> m.step <- stepAIC(m, k=log(nrow(A)), trace=FALSE)
R> summary(m.step)
```

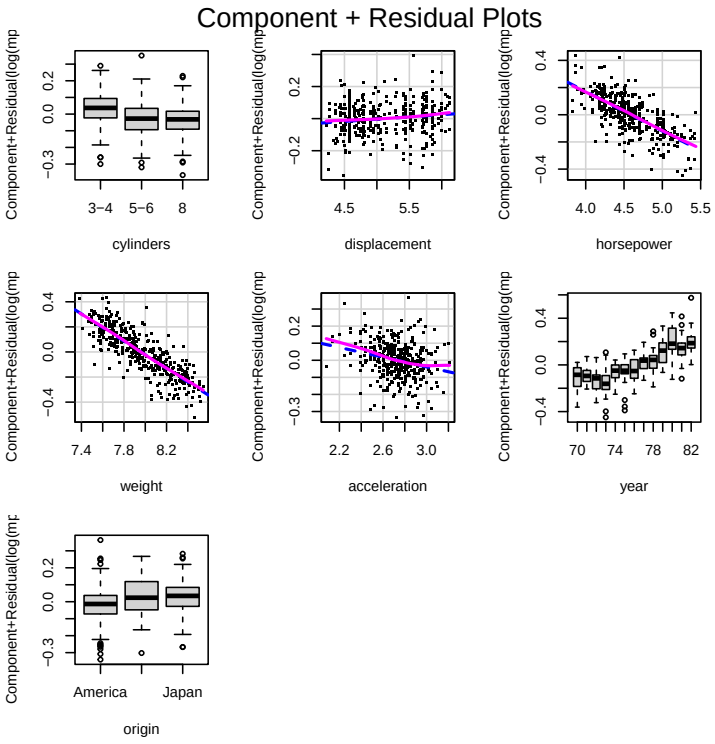


Figure 9: Component+residual plots for the model fit to the transformed `Auto` data

```
Call:
lm(formula = log(mpg) ~ horsepower + weight + acceleration +
    year + origin, data = A)

Residuals:
     Min       1Q   Median       3Q      Max
-0.35230 -0.05682  0.00677  0.06741  0.35861

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   9.434594   0.261529  36.075  < 2e-16
horsepower   -0.276254   0.056143  -4.921 1.30e-06
weight       -0.609071   0.056003 -10.876  < 2e-16
acceleration -0.131380   0.053195  -2.470  0.01397
year71        0.027984   0.028936   0.967  0.33412
year72       -0.007111   0.028446  -0.250  0.80274
year73       -0.039529   0.026014  -1.520  0.12947
year74        0.052752   0.029986   1.759  0.07936
year75        0.053199   0.029280   1.817  0.07004
year76        0.074317   0.028212   2.634  0.00878
year77        0.137931   0.028875   4.777 2.56e-06
year78        0.145876   0.027529   5.299 1.99e-07
```

```
year79        0.236036   0.029080   8.117 6.99e-15
year80        0.335274   0.031148  10.764  < 2e-16
year81        0.262872   0.030555   8.603  < 2e-16
year82        0.323391   0.029608  10.922  < 2e-16
originEurope  0.055818   0.016785   3.326  0.00097
originJapan   0.043554   0.017479   2.492  0.01314

Residual standard error: 0.1049 on 374 degrees of freedom
Multiple R-squared:  0.909, Adjusted R-squared:  0.9049
F-statistic: 219.8 on 17 and 374 DF,  p-value: < 2.2e-16
```

The selected model includes three of the numeric predictors, `horsepower`, `weight`, and `acceleration`, along with the factors `year` and `origin`. We can calculate the MSE for this model, but we expect that the result will be optimistic because we used the whole data to help specify the model:

```
R> mse(Auto$mpg, exp(fitted(m.step)))
```

```
[1] 6.512144
attr(,"casewise loss")
[1] "(y - yhat)^2"
```

This is considerably smaller than the MSE for the original working model:

```
R> mse(Auto$mpg, fitted(m.auto))
```

```
[1] 8.093171
attr(,"casewise loss")
[1] "(y - yhat)^2"
```

A perhaps subtle point is that we compute the MSE for the selected model on the original `mpg` response scale rather than the log scale, so as to make the selected model comparable to the working model.[8]

The `"function"` method for `cv()` allows us to cross-validate the whole model-selection procedure. The first argument to `cv()`—here `selectTransAndStepAIC`—is a model-selection function capable of refitting the model with a fold omitted and returning a CV criterion:

```
R> num.predictors
```

```
[1] "displacement" "horsepower"   "weight"       "acceleration"
```

[8] That's slightly uncomfortable given the skewed distribution of `mpg`. An alternative is to use a robust measure of model lack-of-fit, such as the median absolute error instead of the mean-squared error, employing the `medAbsErr()` function from the **cv** package. The median absolute error, however, cannot be expressed as a casewise average (see Section 7.2).

```
R> cvs <- cv(selectTransStepAIC, data=Auto, seed=76692,
+           working.model=m.auto,
+           predictors=num.predictors,
+           response="mpg", AIC=FALSE)

R RNG seed set to 76692

R> cvs

10-Fold Cross Validation
criterion: mse
cross-validation criterion = 7.485557
bias-adjusted cross-validation criterion = 7.343535
full-sample criterion = 6.512144

R> compareFolds(cvs)

CV criterion by folds:
   fold.1    fold.2    fold.3    fold.4    fold.5    fold.6    fold.7    fold.8
 6.000589 10.509334  7.444420 10.617864  5.049238  7.183160  4.077682 10.501909
   fold.9   fold.10
 9.181184  4.250736

Coefficients by folds:
        (Intercept) horsepower lam.acceleration lam.displacement lam.horsepower
Fold 1      9.71384   -0.17408          0.50000          0.00000        0.00000
Fold 2      9.21713   -0.31480          0.00000          0.00000        0.00000
Fold 3      9.61824   -0.19248          0.00000          0.00000        0.00000
Fold 4      8.69910   -0.25523          0.50000          0.00000        0.00000
Fold 5      9.14403   -0.14934          0.00000          0.00000        0.00000
Fold 6      9.63481   -0.16739          0.50000          0.00000        0.00000
Fold 7      9.98933   -0.36847          0.00000          0.00000       -0.15447
Fold 8      9.06301   -0.29721          0.00000          0.00000        0.00000
Fold 9      8.88315   -0.22684          0.00000          0.00000        0.00000
Fold 10     9.61727   -0.17086          0.00000          0.00000        0.00000
        lam.weight   lambda   weight   year71   year72   year73   year74
Fold 1     0.00000  0.00000 -0.74636  0.03764 -0.00327 -0.02477  0.05606
Fold 2     0.00000  0.00000 -0.47728  0.02173 -0.01488 -0.03770  0.04312
Fold 3     0.00000  0.00000 -0.72085  0.01128 -0.02569 -0.03872  0.05187
Fold 4     0.00000  0.00000 -0.53846  0.02153 -0.02922 -0.05181  0.04136
Fold 5     0.00000  0.00000 -0.69081  0.02531 -0.01062 -0.04625  0.05039
Fold 6     0.00000  0.00000 -0.74049  0.02456  0.00759 -0.03412  0.06266
Fold 7     0.00000  0.00000 -0.72843  0.02532 -0.01271 -0.04144  0.04568
Fold 8     0.00000  0.00000 -0.46392  0.02702 -0.02041 -0.05605  0.04437
Fold 9     0.00000  0.00000 -0.47136  0.00860 -0.03620 -0.04835  0.01906
Fold 10    0.00000  0.00000 -0.73550  0.02937 -0.00899 -0.03814  0.05408
```

```
          year75   year76   year77   year78   year79   year80   year81   year82
Fold 1   0.07080  0.07250  0.14420  0.14281  0.23266  0.35127  0.25635  0.30546
Fold 2   0.04031  0.06718  0.13094  0.14917  0.21871  0.33192  0.26196  0.30943
Fold 3   0.03837  0.06399  0.11593  0.12601  0.20499  0.32821  0.24478  0.29204
Fold 4   0.04072  0.05537  0.12292  0.14083  0.22878  0.32947  0.25140  0.27248
Fold 5   0.05596  0.07044  0.13356  0.14724  0.24675  0.33331  0.26938  0.32594
Fold 6   0.06940  0.07769  0.14211  0.14647  0.23532  0.34761  0.26737  0.33062
Fold 7   0.03614  0.07385  0.12976  0.14040  0.23976  0.33998  0.27652  0.30659
Fold 8   0.06573  0.08135  0.13158  0.13987  0.23011  0.32880  0.25886  0.30538
Fold 9   0.03018  0.05846  0.10536  0.11722  0.20665  0.31533  0.23352  0.29375
Fold 10  0.04881  0.07862  0.14101  0.14313  0.23258  0.35649  0.26214  0.32421
       acceleration displacement cylinders5-6 cylinders8 originEurope
Fold 1
Fold 2      -0.18909     -0.09197
Fold 3
Fold 4      -0.03484                 -0.09080   -0.10909
Fold 5                                                         0.06261
Fold 6
Fold 7
Fold 8      -0.17676     -0.10542
Fold 9      -0.14514     -0.13452
Fold 10
       originJapan
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5         0.04
Fold 6
Fold 7
Fold 8
Fold 9
Fold 10
```

The other arguments to `cv()` are (see `?cv:::cv.function` for additional optional arguments and details):

- `data`, the data set to which the model is fit.
- `seed`, an optional seed for R's pseudo-random-number generator; as for `cv()`, if the seed isn't supplied by the user, then a seed is randomly selected and saved.
- Arguments required by the model-selection function: the starting `working.model` (here, `m.auto`) for transformation and predictor selection; the names of the variables—`predictors` and `response`—that are candidates for transformation; and `AIC=FALSE`, which specifies use of the BIC for model selection.

Some noteworthy points:

- `selectTransStepAIC()` automatically computes CV cost criteria, here the MSE, on the *untransformed* response scale.
- As we anticipated, the estimate of the MSE that we obtain by cross-validating the whole model-specification process is larger than the MSE computed for the model we fit to the `Auto` data separately selecting transformations of the predictors and the response and then selecting predictors for the whole data set.
- When we look at the transformations and predictors selected with each of the 10 folds omitted (i.e., the output of `compareFolds()`), we see that there is little uncertainty in choosing variable transformations (the `lam.*`s for the xs and `lambda` for y in the output), but considerably more uncertainty in subsequently selecting predictors: `horsepower`, `weight`, and `year` are always included among the selected predictors; `acceleration` and `displacement` are included respectively in 4 and 3 of 10 selected models; and `cylinders` and `origin` are each included in only 1 of 10 models. Recall that when we selected predictors for the full data, we obtained a model with `horsepower`, `weight`, `acceleration`, `year`, and `origin`.

4.2. Example: Applying recursive CV to polynomial regression for the Auto data[9]

In Section 2.1, following James *et al.* (2021, Secs. 5.1, 5.3), we fit polynomial regressions up to degree 10 to the relationship of `mpg` to `horsepower` for the `Auto` data, saving the 10 models, named `m.1` through `m.10`, in the list `mlist`. We then used `cv()` to compare the cross-validated MSE for the 10 models, discovering that the 7th degree polynomial had the smallest MSE (by a small margin).

If we select the 7th degree polynomial model, intending to use it for prediction, the CV estimate of the MSE for this model will be optimistic. One solution is to cross-validate the process of using CV to select the "best" model—that is, to apply CV to CV recursively. The function `selectModelList()`, which is suitable for use with `cv()`, implements this idea.

Applying `selectModelList()` to the `Auto` polynomial-regression models, and using 10-fold CV, we obtain:

```
R> recursiveCV.auto <- cv(selectModelList, data=Auto,
+                         working.model=mlist,
+                         save.model=TRUE, seed=2120)

R RNG seed set to 2120

R> recursiveCV.auto

10-Fold Cross Validation
criterion: mse
cross-validation criterion = 20.01249
bias-adjusted cross-validation criterion = 20.61866
full-sample criterion = 18.74615
```

[9]What we call here "recursive CV" has also been termed "nested CV" (e.g., in the **cvms** package by Olsen and Zachariae 2024). We prefer, however, to reserve the term "nested CV" for the procedure described by Bates, Hastie, and Tibshirani (2023).

```
R> recursiveCV.auto$selected.model


Call:
lm(formula = mpg ~ poly(horsepower, p), data = Auto)

Coefficients:
         (Intercept)  poly(horsepower, p)1  poly(horsepower, p)2
              23.446              -120.138                44.090
poly(horsepower, p)3  poly(horsepower, p)4  poly(horsepower, p)5
              -3.949                -5.188                13.272
poly(horsepower, p)6  poly(horsepower, p)7
              -8.546                 7.981

R> cv(mlist[[7]], seed=2120) # same seed for same folds

R RNG seed set to 2120

10-Fold Cross Validation
method: Woodbury
criterion: mse
cross-validation criterion = 18.89822
bias-adjusted cross-validation criterion = 18.85429
full-sample criterion = 18.07817
```

As expected, recursive CV produces a larger estimate of MSE for the selected 7th degree polynomial model than CV applied directly to this model.

We can equivalently call `cv()` with the list of models as its first argument and set the argument `recursive=TRUE`:

```
R> cv(mlist, data=Auto, seed=2120, recursive=TRUE, save.model=TRUE)

R RNG seed set to 2120

10-Fold Cross Validation
cross-validation criterion = 20.01249
bias-adjusted cross-validation criterion = 20.61866
full-sample criterion = 18.74615
```

5. Extending the cv package

The **cv** package is designed to be extensible in several directions. In order of increasing general complexity, we can add: (1) a cross-validation cost criterion; (2) a model class that's not directly accommodated by the `cv()` default method or by another directly inherited method; and (3) a new model-selection procedure suitable for use with the `"function"` method for

`cv()`. In this section, we illustrate (1) and (2); more diverse and extensive examples may be found in the vignette on extending the **cv** package.

Suppose that we want to cross-validate a multinomial logistic regression model fit by the `multinom()` function in the **nnet** package (Venables and Ripley 2002). We borrow an example from Fox (2016, Sec. 14.2.1), with data from the British Election Panel Study on vote choice in the 2001 British election. Data for the example are in the `BEPS` data set in the **carData** package:

```
R> data("BEPS", package="carData")
R> summary(BEPS)
```

```
               vote          age        economic.cond.national
 Conservative    :462   Min.   :24.00   Min.   :1.000
 Labour          :720   1st Qu.:41.00   1st Qu.:3.000
 Liberal Democrat:343   Median :53.00   Median :3.000
                        Mean   :54.18   Mean   :3.246
                        3rd Qu.:67.00   3rd Qu.:4.000
                        Max.   :93.00   Max.   :5.000
 economic.cond.household     Blair           Hague          Kennedy
 Min.   :1.00            Min.   :1.000   Min.   :1.000   Min.   :1.000
 1st Qu.:3.00            1st Qu.:2.000   1st Qu.:2.000   1st Qu.:2.000
 Median :3.00            Median :4.000   Median :2.000   Median :3.000
 Mean   :3.14            Mean   :3.334   Mean   :2.747   Mean   :3.135
 3rd Qu.:4.00            3rd Qu.:4.000   3rd Qu.:4.000   3rd Qu.:4.000
 Max.   :5.00            Max.   :5.000   Max.   :5.000   Max.   :5.000
     Europe       political.knowledge    gender
 Min.   : 1.000   Min.   :0.000       female:812
 1st Qu.: 4.000   1st Qu.:0.000       male  :713
 Median : 6.000   Median :2.000
 Mean   : 6.729   Mean   :1.542
 3rd Qu.:10.000   3rd Qu.:2.000
 Max.   :11.000   Max.   :3.000
```

The polytomous (multi-category) response variable is `vote`, a factor with levels `"Conservative"`, `"Labour"`, and `"Liberal Democrat"`. The predictors of `vote` are:

- `age`, in years;
- `econ.cond.national` and `econ.cond.household`, the respondent's ratings of the state of the economy, on 1 to 5 scales.
- `Blair`, `Hague`, and `Kennedy`, ratings of the leaders of the Labour, Conservative, and Liberal Democratic parties, on 1 to 5 scales.
- `Europe`, an 11-point scale on attitude towards European integration, with high scores representing "Euro-skepticism."
- `political.knowledge`, knowledge of the parties' positions on European integration, with scores from 0 to 3.
- `gender`, `"female"` or `"male"`.

The model fit to the data includes an interaction between `Europe` and `political.knowledge`, which was the focus of the original research on which this example is based (Andersen, Heath, and Sinnott 2002); the other predictors enter the model additively:

```
R> library("nnet", quietly=TRUE)
R> m.beps <- multinom(vote ~ age + gender + economic.cond.national +
+                         economic.cond.household + Blair + Hague + Kennedy +
+                         Europe*political.knowledge, data=BEPS)

# weights:  36 (22 variable)
initial  value 1675.383740
iter  10 value 1240.047788
iter  20 value 1163.199642
iter  30 value 1116.519687
final  value 1116.519666
converged
```

The `Europe` × `political.knowledge` interaction is associated with a very small p-value. Figure 10 shows an "effect plot," using the the **effects** package (Fox and Weisberg 2019) to visualize the interaction in a "stacked-area" graph:

```
R> plot(effects::Effect(c("Europe", "political.knowledge"), m.beps,
+             xlevels=list(Europe=1:11, political.knowledge=0:3),
+             fixed.predictors=list(given.values=c(gendermale=0.5))),
+      lines=list(col=c("blue", "red", "orange")), # party colors
+      axes=list(x=list(rug=FALSE), y=list(style="stacked")))
```

As political knowledge increases, voters tend to align their votes more closely with the party positions on European integration: The Conservative Party was relatively Euro-skeptic, while Labour and the Liberal Democrats were more supportive of the UK's participation in the EU.

To cross-validate this multinomial-logit model we need an appropriate cost criterion. None of the criteria in the **cv** package will do—for example, `mse()` is appropriate only for a numeric response. The `BayesRule()` criterion, also supplied by **cv**, which is for a binary response, comes close:

```
R> BayesRule

function(y, yhat) {
  if (!all(y %in% c(0, 1)))
    stop("response values not all 0 or 1")
  if (any(yhat < 0) ||
      any(yhat > 1))
    stop("fitted values outside of interval [0, 1]")
  yhat <- round(yhat)
  result <- mean(y != yhat) # proportion in error
  attr(result, "casewise loss") <- "y != round(yhat)"
```

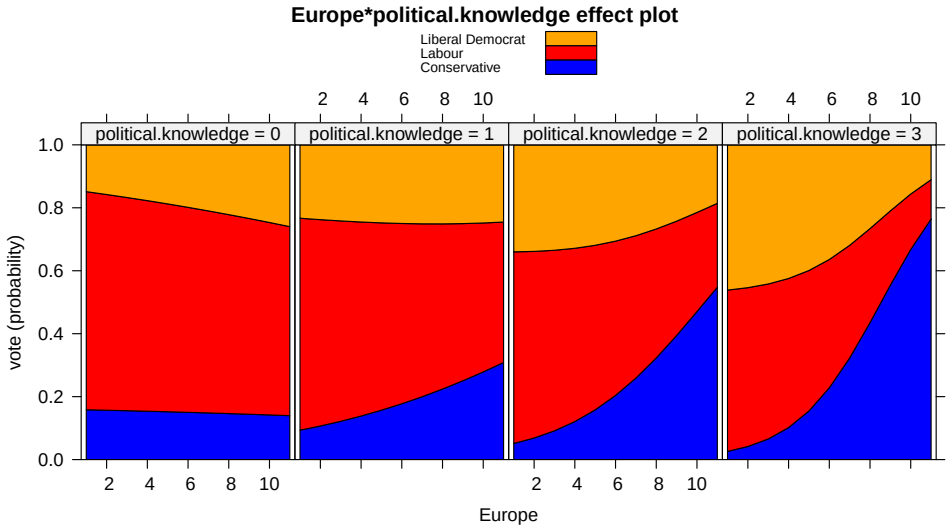


Figure 10: Effect plot for the interaction between attitude towards European integration and political knowledge in the multinomial logit model fit to voting data from the 2001 British Election Panel Study.

```
  result
}
<bytecode: 0x142fa0f70>
<environment: namespace:cv>
```

After doing some error checking, `BayesRule()` rounds the predicted proabability of a 1 ("success") response in a binary regression model to 0 or 1 to obtain a categorical prediction, and then reports the proportion of incorrect predictions. Because the Bayes's rule criterion is an average of casewise components (as, e.g., is the MSE), a `"casewise loss"` attribute is attached to the result, enabling the computation of bias correction and confidence intervals (as discussed in Section 7.2).

It is straightforward to adapt Bayes's rule to a polytomous response:

```
R> head(BEPS$vote)

[1] Liberal Democrat Labour           Labour           Labour
[5] Labour           Labour
Levels: Conservative Labour Liberal Democrat

R> yhat <- predict(m.beps, type="class")
R> head(yhat)

[1] Labour           Labour           Labour           Labour
[5] Liberal Democrat Labour
Levels: Conservative Labour Liberal Democrat

R> BayesRuleMulti <- function(y, yhat){
+   result <- mean(y != yhat)
```

```
+   attr(result, "casewise loss") <- "y != yhat"
+   result
+ }
R>
R> BayesRuleMulti(BEPS$vote, yhat)

[1] 0.3186885
attr(,"casewise loss")
[1] "y != yhat"
```

The `predict()` method for `"multinom"` models called with argument `type="class"` reports the Bayes's rule prediction for each case—that is, the response category with the highest predicted probability. Our `BayesRuleMulti()` function calculates the proportion of misclassified cases. Because this value is also the mean of casewise components, we attach a `"casewise loss"` attribute to the result.

The marginal proportions for the response categories are

```
R> xtabs(~ vote, data=BEPS)/nrow(BEPS)

vote
    Conservative           Labour Liberal Democrat
       0.3029508        0.4721311        0.2249180
```

and so the marginal Bayes's rule prediction, that everyone will vote Labour, produces an error rate of $1 - 0.47213 = 0.52787$. The multinomial-logit model appears to do substantially better than that, but does its performance hold up to cross-validation?

We check first whether the default `cv()` method works "out-of-the-box" for the `"multinom"` model:

```
R> cv(m.beps, seed=3465, criterion=BayesRuleMulti)

Error in GetResponse.default(model): non-vector response
```

The default method of `GetResponse()` (a function supplied by the **cv** package—see `?GetResponse`) fails for a `"multinom"` object. A straightforward solution is to supply a `GetResponse.multinom()` method that returns the factor response (using the `get_response()` function from the **insight** package, Lüdecke, Waggoner, and Makowski 2019),

```
R> GetResponse.multinom <- function(model, ...) {
+   insight::get_response(model)
+ }
R>
R> head(GetResponse(m.beps))

[1] Liberal Democrat Labour           Labour           Labour
[5] Labour           Labour
Levels: Conservative Labour Liberal Democrat
```

and to try again:

```
R> cv(m.beps, seed=3465, criterion=BayesRuleMulti)

R RNG seed set to 3465

# weights:  36 (22 variable)
initial  value 1507.296060
iter  10 value 1134.575036
iter  20 value 1037.413231
iter  30 value 1007.705242
iter  30 value 1007.705235
iter  30 value 1007.705235
final  value 1007.705235
converged

Error in match.arg(type): 'arg' should be one of "class", "probs"
```

A `traceback()` (not shown) reveals that the problem is that the default method of `cv()` calls the `"multinom"` method for `predict()` with the argument `type="response"`, when the correct argument should be `type="class"`. We therefore must write a `"multinom"` method for `cv()`, but that proves to be very simple:

```
R> cv.multinom <- function (model, data, criterion = BayesRuleMulti,
+                           k, reps, seed, ...) {
+     model <- update(model, trace = FALSE)
+     NextMethod(
+       type = "class",
+       criterion = criterion,
+       criterion.name = deparse(substitute(criterion))
+     )
+   }
```

That is, we simply call the default `cv()` method (via `NextMethod()`) with the `type` argument properly set. In addition to supplying the correct `type` argument, our method sets the default `criterion` for the `cv.multinom()` method to `BayesRuleMulti`. Adding the argument `criterion.name= deparse(substitute(criterion))` is inessential, but it insures that printed output will include the name of the criterion function that's employed, whether it's the default `BayesRuleMulti` or something else. Prior to invoking `NextMethod()`, we called `update()` with `trace=FALSE` to suppress the iteration history reported by default by `multinom()`—it would be tedious to see the iteration history for each fold.

Then:

```
R> cv(m.beps, seed=3465)

R RNG seed set to 3465
```

```
10-Fold Cross Validation
criterion: BayesRuleMulti
cross-validation criterion = 0.3245902
bias-adjusted cross-validation criterion = 0.3236756
95% CI for bias-adjusted CV criterion = (0.300168, 0.3471831)
full-sample criterion = 0.3186885
```

The cross-validated polytomous Bayes's rule criterion confirms that the fitted model does substantially better than the marginal Bayes's rule prediction that everyone votes for Labour.

6. Comparing cv to other R software for cross-validation

The **cv** package is far from unique in implementing cross-validation of regression models in R. We've already mentioned the `cv.glm()` function in the **boot** package. A general review of CV facilities in R is beyond the scope of the current paper, but in this section we remark selectively on other R packages that support CV.

Our goal in implementing CV is to make it conveniently available to R users with limited programming skills,[10] and hence to encourage its use. Towards this end, the **cv** package provides a simple, consistent interface through the `cv()` generic function, with specific methods for different classes of standard R statistical models. Moreover, we have tried to make writing methods for additional classes of statistical models as simple and straightforward as possible. Because CV is computationally intensive, we also aim for efficiency, by enabling parallel computations generally, and by exploiting properties of specific classes of statistical models (in particular, computations based on hatvalues and the Woodbury matrix identity for linear and generalized linear model). Finally, the package includes some unusual features, such as general support for mixed-effects models and for cross-validating complex model-specification procedures.

Some R packages for statistical learning include facilities for cross-validation. Notable examples are the **caret** package (Kuhn 2008) and the **tidyfit** package (Pfitzinger 2024); the latter employs **rsample** (Frick, Chow, Kuhn, Mahoney, Silge, and Wickham 2024) for CV, which we will discuss presently. This contrasts with the approach taken in the **cv** package, which, as we have explained, directly supports classes of standard R statistical models.

Other packages, including **cvms** (Olsen and Zachariae 2024), **mlexperiments** (Kapsner 2024), **origami** (Coyle, Hejazi, Malenica, and Phillips 2022), and **rsample** (which also supports bootstrapping), modularize the CV process to a greater or lesser extent. Advantages of this approach include flexibility and generality, but a disadvantage is that use of these packages entails nontrivial programming effort and skill.

We illustrate with an example adapted from a vignette in the **rsample** package, which uses the `attrition` data set from the **modeldata** package (Kuhn 2024) (and which is by a wide margin the most downloaded of the CV packages mentioned in the preceding paragraph). Here, with minimal explanation, is how one would perform LOO CV for this example using

[10] R users with sophisticated programming skills would generally find it unchallenging to implement CV directly for specific applications. Even in this case, however, there's an argument for using a simple pre-programmed interface to CV, to minimize programming effort and to avoid mistakes.

rsample:[11]

```
R> library(rsample)
R>
R> data("attrition", package = "modeldata")
R> nrow(attrition)

[1] 1470

R> # formula for model to be fit; Attrition is binary:
R> mod_form <- Attrition ~ JobSatisfaction + Gender + MonthlyIncome
R>
R> rs_obj <- loo_cv(attrition) # original uses vfold_cv()
R>
R> # user-supplied function:
R> holdout_results <- function(splits, ...) {
+   mod <- glm(..., data=analysis(splits), family=binomial)
+   holdout <- assessment(splits)
+   res <- broom::augment(mod, newdata=holdout)
+   lvls <- levels(holdout$Attrition)
+   predictions <- factor(ifelse(res$.fitted > 0, lvls[2], lvls[1]),
+                         levels=lvls)
+   res$correct <- predictions == holdout$Attrition
+   res
+ }
R>
R> library(purrr, warn.conflicts=FALSE)
R> # apply CV:
R> rs_obj$results <- map(rs_obj$splits, holdout_results, mod_form)
R> # summarize results:
R> rs_obj$accuracy <- map_dbl(rs_obj$results,
+                             function(x) mean(x$correct))
R> # accuracy is the complement of the Bayes-rule CV criterion:
R> 1 - summary(rs_obj$accuracy)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 1.0000  0.0000  0.0000  0.1612  0.0000  0.0000
```

And here's how we can perform the same computation using the `cv()` function in the **cv** package:

```
R> mod.attrition <- glm(mod_form, data=attrition, family=binomial)
R> (cv.attrition <- cv(mod.attrition, k="loo", criterion=BayesRule))
```

[11]The example in the **rsample** vignette uses 10-fold rather than LOO CV, but otherwise the code from the vignette is only trivially modified here.

```
n-Fold Cross Validation
method: exact
criterion: BayesRule
cross-validation criterion = 0.1612245
bias-adjusted cross-validation criterion = 0.1612245
95% CI for bias-adjusted CV criterion = (0.1424194, 0.1800296)
full-sample criterion = 0.1612245
```

```
R> all.equal(cv.attrition$`CV crit`, 1 - mean(rs_obj$accuracy),
+           check.attributes=FALSE)
```

```
[1] TRUE
```

Let's compare the computational efficiency of the two implementations, also showing the use of `method="Woodbury"` and `method="hatvalues"` for `cv()`, which, in this example, produce numerically identical results to the default `method="exact"`:

```
R> all.equal(cv.attrition$`CV crit`,
+           cv(mod.attrition, k="loo", criterion=BayesRule,
+              method="Woodbury")$`CV crit`,
+           )
```

```
[1] TRUE
```

```
R> all.equal(cv.attrition$`CV crit`,
+           cv(mod.attrition, k="loo", criterion=BayesRule,
+              method="hatvalues")$`CV crit`
+           )
```

```
[1] TRUE
```

```
R> microbenchmark::microbenchmark(
+   rsample={
+     rs_obj$results <- map(rs_obj$splits, holdout_results, mod_form)
+     rs_obj$accuracy <- map_dbl(rs_obj$results,
+                                function(x) mean(x$correct))
+   },
+   cv.exact=cv(mod.attrition, k="loo", criterion=BayesRule),
+   cv.wood=cv(mod.attrition, k="loo", criterion=BayesRule,
+              method="Woodbury"),
+   cv.hatvalues=cv(mod.attrition, k="loo", criterion=BayesRule,
+                   method="hatvalues"),
+   times=10
+ )
```

```
Unit: milliseconds
         expr         min          lq        mean      median          uq
      rsample 5858.845429 5918.511745 6182.166132 6032.719008 6513.721496
    cv.exact 4823.821011 4879.622175 4986.028516 4973.975331 5051.786505
     cv.wood  103.815690  107.823973  112.389204  112.138341  117.148398
 cv.hatvalues    1.248655    1.299823    1.540665    1.587007    1.654432
         max neval
 6707.589832    10
 5305.047728    10
  121.483287    10
    1.916832    10
```

Computing time for **rsample** for this problem is similar to `cv()` with `method="exact"`, but more than an order of magnitude slower than `cv()` with `method="Woodbury"`, and more than three orders of magnitude slower then `cv()` with `method="hatvalues"`.

7. Computational notes

7.1. Efficient computations for linear and generalized linear models

The most straightforward way to implement cross-validation in R for statistical modeling functions that are written in the canonical manner is to use `update()` to refit the model with each fold removed. This is the approach taken in the default method for `cv()`, and it is appropriate if the cases are independently sampled. Refitting the model in this manner for each fold is generally feasible when the number of folds in modest, but can be prohibitively costly for leave-one-out cross-validation when the number of cases is large.

The `"lm"` and `"glm"` methods for `cv()` take advantage of computational efficiencies by avoiding refitting the model with each fold removed. Consider, in particular, the weighted linear model $\mathbf{y}_{n\times 1} = \mathbf{X}_{n\times p}\boldsymbol{\beta}_{p\times 1} + \boldsymbol{\varepsilon}_{n\times 1}$, where $\boldsymbol{\varepsilon} \sim \mathbf{N}_n\left(\mathbf{0}, \sigma^2\mathbf{W}^{-1}_{n\times n}\right)$. Here, $\mathbf{y}$ is the response vector, $\mathbf{X}$ the model matrix, and $\boldsymbol{\varepsilon}$ the error vector, each for n cases, and $\boldsymbol{\beta}$ is the vector of p population regression coefficients. The errors are assumed to be multivariately normally distributed with 0 means and covariance matrix $\sigma^2\mathbf{W}^{-1}$, where $\mathbf{W} = \text{diag}(w_i)$ is a diagonal matrix of inverse-variance weights. For the linear model with constant error variance, the weight matrix is taken to be $\mathbf{W} = \mathbf{I}_n$, the order-$n$ identity matrix.

The weighted-least-squares (WLS) estimator of $\boldsymbol{\beta}$ is (see, e.g., Fox 2016, Sec. 12.2.2) [12]

$$\mathbf{b}_{\text{WLS}} = \left(\mathbf{X}^T\mathbf{W}\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{W}\mathbf{y}$$

Fitted values are then $\widehat{\mathbf{y}} = \mathbf{X}\mathbf{b}_{\text{WLS}}$.

The LOO fitted value for the ith case can be efficiently computed by $\widehat{y}_{-i} = y_i - e_i/(1-h_i)$ where $h_i = \mathbf{x}_i^T\left(\mathbf{X}^T\mathbf{W}\mathbf{X}\right)^{-1}\mathbf{x}_i$ (the so-called "hatvalue"). Here, $\mathbf{x}_i^T$ is the ith row of $\mathbf{X}$, and

[12] This is a definitional formula, which assumes that the model matrix $\mathbf{X}$ is of full column rank, and which can be subject to numerical instability when $\mathbf{X}$ is ill-conditioned. `lm()` uses the singular-value decomposition of the model matrix to obtain computationally more stable results.

$\mathbf{x}_i$ is the ith row written as a column vector. This approach can break down when one or more hatvalues are equal to 1, in which case the formula for $\widehat{y}_{-i}$ requires division by 0. In this case, the "training" set omitting the observation with hatvalue = 1 is rank-deficient and the predictors for the left-out case are outside the linear span of the predictors in the training set.

To compute cross-validated fitted values when the folds contain more than one case, we make use of the Woodbury matrix identify (Hager 1989),

$$(\mathbf{A}_{m\times m} + \mathbf{U}_{m\times k}\mathbf{C}_{k\times k}\mathbf{V}_{k\times m})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}\left(\mathbf{C}^{-1} + \mathbf{V}\mathbf{A}^{-1}\mathbf{U}\right)^{-1}\mathbf{V}\mathbf{A}^{-1}$$

where $\mathbf{A}$ is a nonsingular order-n matrix. We apply this result by letting

$$\begin{aligned}\mathbf{A} &= \mathbf{X}^T\mathbf{W}\mathbf{X}\\ \mathbf{U} &= \mathbf{X}_\mathbf{j}^T\\ \mathbf{V} &= -\mathbf{X}_\mathbf{j}\\ \mathbf{C} &= \mathbf{W}_\mathbf{j}\end{aligned}$$

where the subscript $\mathbf{j} = (i_{j1}, \ldots, i_{jm})^T$ represents the vector of indices for the cases in the jth fold, $j = 1, \ldots, k$. The negative sign in $\mathbf{V} = -\mathbf{X}_\mathbf{j}$ reflects the *removal*, rather than addition, of the cases in $\mathbf{j}$.

Applying the Woodbury identity isn't quite as fast as using the hatvalues, but it is generally much faster than refitting the model. A disadvantage of the Woodbury identity, however, is that it entails explicit matrix inversion and thus may be numerically unstable. The inverse of $\mathbf{A} = \mathbf{X}^T\mathbf{W}\mathbf{X}$ is available directly in the `"lm"` object, but the second term on the right-hand side of the Woodbury identity requires a matrix inversion with each fold deleted. (In contrast, the inverse of each $\mathbf{C} = \mathbf{W}_\mathbf{j}$ is straightforward because $\mathbf{W}$ is diagonal.)

The Woodbury identity also requires that the model matrix be of full rank. We impose that restriction in our code by removing redundant regressors from the model matrix for all of the cases, but that doesn't preclude rank deficiency from surfacing when a fold is removed. Rank deficiency of $\mathbf{X}$ doesn't disqualify cross-validation because all we need are fitted values under the estimated model.

`glm()` computes the maximum-likelihood estimates for a generalized linear model by iterated weighted least squares (see, e.g., Fox and Weisberg 2019, Sec. 6.12). The last iteration is therefore just a WLS fit of the "working response" on the model matrix using "working weights." Both the working weights and the working response at convergence are available from the information in the object returned by `glm()`.

We then treat re-estimation of the model with a case or cases deleted as a WLS problem, using the hatvalues or the Woodbury matrix identity. The resulting fitted values for the deleted fold aren't exact—that is, except for the Gaussian family, the result isn't identical to what we would obtain by literally refitting the model—but in our (limited) experience, the approximation is very good, especially for LOO CV, which is when we would be most tempted to use it. Nevertheless, because these results are approximate, the default for the `"glm"` `cv()` method is to perform the exact computation, which entails refitting the model with each fold omitted.

7.2. Computation of the bias-corrected CV criterion and confidence intervals

Let $\mathrm{CV}(\mathbf{y}, \widehat{\mathbf{y}})$ represent a cross-validation cost criterion, such as mean-squared error, computed for all of the n values of the response $\mathbf{y}$ based on fitted values $\widehat{\mathbf{y}}$ from the model fit to all of the data. We require that $\mathrm{CV}(\mathbf{y}, \widehat{\mathbf{y}})$ is the mean of casewise components, that is, $\mathrm{CV}(\mathbf{y}, \widehat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n} \mathrm{cv}(y_i, \widehat{y}_i)$.[13] For example, $\mathrm{MSE}(\mathbf{y}, \widehat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2$.[14]

We divide the n cases into k folds of approximately $n_j \approx n/k$ cases each, where $n = \sum n_j$. As above, let $\mathbf{j}$ denote the indices of the cases in the jth fold.

Now define $\mathrm{CV}_j = \mathrm{CV}(\mathbf{y}, \widehat{\mathbf{y}}^{(j)})$. The superscript (j) on $\widehat{\mathbf{y}}^{(j)}$ represents fitted values computed for all of the cases from the model with fold j omitted. Let $\widehat{\mathbf{y}}^{(-i)}$ represent the vector of fitted values for all n cases where the fitted value for the ith case is computed from the model fit with the fold including the ith case omitted (i.e., fold j for which $i \in \mathbf{j}$).

Then the cross-validation criterion is just $\mathrm{CV} = \mathrm{CV}(\mathbf{y}, \widehat{\mathbf{y}}^{(-i)})$. Following Davison and Hinkley (1997, pp. 293–295), the bias-adjusted cross-validation criterion is

$$\mathrm{CV}_{\mathrm{adj}} = \mathrm{CV} + \mathrm{CV}(\mathbf{y}, \widehat{\mathbf{y}}) - \frac{1}{n}\sum_{j=1}^{k} n_j \mathrm{CV}_j$$

We compute the standard error of CV as

$$\mathrm{SE}(\mathrm{CV}) = \frac{1}{\sqrt{n}}\sqrt{\frac{\sum_{i=1}^{n}\left[\mathrm{cv}(y_i, \widehat{y}_i^{(-i)}) - \mathrm{CV}\right]^2}{n-1}}$$

that is, as the standard deviation of the casewise components of CV divided by the square-root of the number of cases.

We then use SE(CV) to construct a $100 \times (1-\alpha)\%$ confidence interval around the *adjusted* CV estimate of error:

$$\left[\mathrm{CV}_{\mathrm{adj}} - z_{1-\alpha/2}\mathrm{SE}(\mathrm{CV}), \mathrm{CV}_{\mathrm{adj}} + z_{1-\alpha/2}\mathrm{SE}(\mathrm{CV})\right]$$

where $z_{1-\alpha/2}$ is the $1-\alpha/2$ quantile of the standard-normal distribution (e.g, $z \approx 1.96$ for a 95% confidence interval, for which $1-\alpha/2 = .975$).

Bates *et al.* (2023) show that the coverage of this confidence interval is poor for small samples, and they suggest a much more computationally intensive procedure, called *nested cross-validation*, to compute better estimates of error and confidence intervals with better coverage for small samples. We may implement Bates et al.'s approach in a later release of the **cv** package. At present we use the confidence interval above for sufficiently large n, which, based on Bates et al.'s results, we take by default to be $n \geq 400$.

References

[13] Arlot and Celisse (2010) term the casewise loss, $\mathrm{cv}(y_i, \widehat{y}_i)$, the "contrast function."

[14] Some commonly employed CV criteria—such as the root-mean-squared error (RMSE), median absolute error, and, for binary-regression models, the complement of the area under the receiver operating characteristic (ROC) curve—are not means of casewise components. That the RMSE and median absolute error aren't means of casewise components is obvious; for the complement of the area under the ROC curve, see the vignette on extending the **cv** package.

Andersen R, Heath A, Sinnott R (2002). "Political knowledge and electoral choice." *British Elections and Parties Review*, **12**, 11–27.

Arlot S, Celisse A (2010). "A survey of cross-validation procedures for model selection." *Statistics Surveys*, **4**, 40 – 79. URL https://doi.org/10.1214/09-SS054.

Bates D, Mächler M, Bolker B, Walker S (2015). "Fitting Linear Mixed-Effects Models Using lme4." *Journal of Statistical Software*, **67**(1), 1–48.

Bates S, Hastie T, Tibshirani R (2023). "Cross-validation: What does it estimate and how well does it do it?" *Journal of the American Statistical Association*, **in press**. URL https://doi.org/10.1080/01621459.2023.2197686.

Box GEP, Cox DR (1964). "An analysis of transformations." *Journal of the Royal Statistical Society, Series B*, **26**, 211–252.

Brooks ME, Kristensen K, van Benthem KJ, Magnusson A, Berg CW, Nielsen A, Skaug HJ, Maechler M, Bolker BM (2017). "glmmTMB Balances Speed and Flexibility Among Packages for Zero-inflated Generalized Linear Mixed Modeling." *The R Journal*, **9**(2), 378–400. doi:10.32614/RJ-2017-066.

Canty A, Ripley BD (2022). *boot: Bootstrap R (S-Plus) Functions*. R package version 1.3-28.1.

Coyle J, Hejazi N, Malenica I, Phillips R (2022). *origami: Generalized Framework for Cross-Validation*. R package version 1.0.7, URL https://CRAN.R-project.org/package=origami.

Davison AC, Hinkley DV (1997). *Bootstrap Methods and Their Applications*. Cambridge University Press, Cambridge.

Fox J (2016). *Applied Regression Analysis and Generalized Linear Models*. Second edition edition. Sage, Thousand Oaks CA.

Fox J, Weisberg S (2019). *An R Companion to Applied Regression*. Third edition edition. Sage, Thousand Oaks CA.

Frick H, Chow F, Kuhn M, Mahoney M, Silge J, Wickham H (2024). *rsample: General Resampling Infrastructure*. R package version 1.2.1, URL https://CRAN.R-project.org/package=rsample.

Hager WW (1989). "Updating the inverse of a matrix." *SIAM Review*, **31**(2), 221–239.

Harrell Jr F (2015). *Regression Modeling Strategies*. Second edition edition. Springer, New York.

Hastie T, Tibshirani R, Friedman J (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Second edition edition. Springer, New York. URL https://hastie.su.domains/ElemStatLearn/printings/ESLII_print12_toc.pdf.

James G, Witten D, Hastie T, Tibshirani R (2021). *An Introduction to Statistical Learning with Applications in R*. Second edition edition. Springer, New York.

Kapsner LA (2024). *mlexperiments: Machine Learning Experiments*. R package version 0.0.3, URL https://CRAN.R-project.org/package=mlexperiments.

Kuhn M (2008). "Building Predictive Models in R Using the caret Package." *Journal of Statistical Software*, **28**(5), 1–26. doi:10.18637/jss.v028.i05. URL https://www.jstatsoft.org/index.php/jss/article/view/v028i05.

Kuhn M (2024). *modeldata: Data Sets Useful for Modeling Examples*. R package version 1.3.0, URL https://CRAN.R-project.org/package=modeldata.

Lüdecke D, Waggoner P, Makowski D (2019). "insight: A Unified Interface to Access Information from Model Objects in R." *Journal of Open Source Software*, **4**(38), 1412.

Mersmann O (2023). *microbenchmark: Accurate Timing Functions*. R package version 1.4.10, URL https://CRAN.R-project.org/package=microbenchmark.

Olsen LR, Zachariae HB (2024). *cvms: Cross-Validation for Model Selection*. R package version 1.6.1, URL https://CRAN.R-project.org/package=cvms.

Pfitzinger J (2024). *tidyfit: Regularized Linear Modeling with Tidy Data*. R package version 0.7.1, URL https://CRAN.R-project.org/package=tidyfit.

Pinheiro JC, Bates DM (2000). *Mixed-Effects Models in S and S-PLUS*. Springer, New York.

Raudenbush SW, Bryk AS (2002). *Hierarchical Linear Models: Applications and data analysis methods*. Second edition edition. Sage, Thousand Oaks CA.

Vehtari A (2023). "Cross-validation FAQ." URL https://users.aalto.fi/~ave/CV-FAQ.html.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*. Fourth edition edition. Springer, New York.

Weisberg S (2014). *Applied Linear Regression*. Second edition edition. Wiley, Hoboken NJ.

Affiliation:

John Fox
McMaster University
Hamilton, Ontario, Canada
E-mail: jfox@mcmaster.ca
URL: https://www.john-fox.ca/

Georges Monette
York University
Toronto, Ontario, Canada

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd