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 can also 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 (see, e.g., Barnard 1974): The data are randomly divided into two, possibly equal-size, parts; the first part is used to develop and fit (in the language of machine leaning, "train") a statistical model; and then the second part is used to assess the adequacy of ("validate") 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 cogent 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 equals 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 the prediction error compared with a model fit to n cases. The n regression models are highly statistically dependent, however, based as they are on nearly the same data, and so the resulting estimate of prediction error has larger variance than if the predictions from the models fit to the n data sets were independent.

Because predictions are based on smaller data sets, each of size approximately $n - n/k$, estimated prediction error for k-fold CV with $k = 5$ or 10 (commonly employed choices) is more biased than estimated prediction error for LOO CV. It is possible, however, to correct k-fold CV for bias (see Appendix A.2).

The relative *variance* of prediction error for LOO CV and k-fold CV (with $k < n$) is more complicated: Because the overlap between the data sets with each fold omitted is smaller for k-fold CV than for LOO CV, the dependencies among the predictions are smaller for the former than for the latter, tending to produce smaller variance in prediction error for k-fold CV. In contrast, there are factors that tend to inflate the variance of prediction error in k-fold CV, including the reduced size of the data sets with each fold omitted and the randomness induced by the selection of folds—in LOO CV the folds are not random.

The **cv** package for R automates the process of cross-validation for standard R statistical models. The principal function in the package, 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 `cv()` function is introduced in the context of a preliminary example in Section 2 of the paper.

- 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.
- 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 potentially 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. In Section 6, we briefly situate the **cv** package relative to other R software for cross-validation (directly comparing **cv** to the **rsample** and **caret** packages), and to other statistical software (in particular, SAS, Stata, and Python).

An appendix to the paper 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 provide greater detail on some topics that we do describe, such as extensions to the package.

2. Preliminary example: Polynomial regression

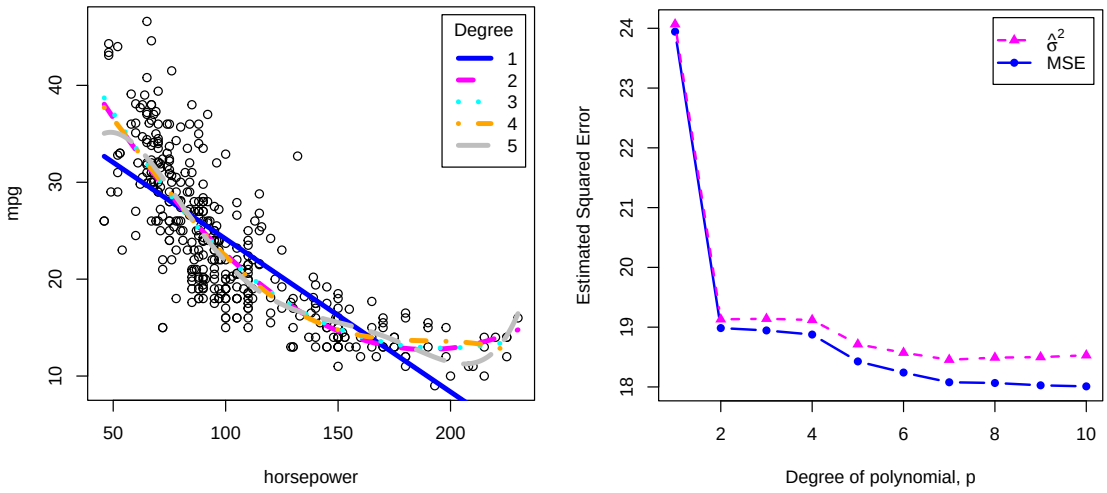
The data for the example in this section are drawn from the **ISLR2** package for R, associated with James *et al.* (2021). The presentation here is close (though not identical) to that in the original source (James *et al.* 2021, Secs. 5.1, 5.3), and it demonstrates the use of the `cv()` function.[1]

The `Auto` data set contains information about 392 cars:

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

```
      mpg          cylinders      displacement    horsepower        weight
 Min.   : 9.0   Min.   :3.00   Min.   : 68   Min.   : 46.0   Min.   :1613
```

[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.



(a) (b)

Figure 1: (a) `mpg` vs. `horsepower` for the `Auto` data, showing fitted polynomials of degree 1 through 5. (b) Estimated squared error as a function of polynomial degree, $p = 1, \ldots, 10$.

```
1st Qu.:17.0   1st Qu.:4.00   1st Qu.:105   1st Qu.: 75.0   1st Qu.:2225
Median :22.8   Median :4.00   Median :151   Median : 93.5   Median :2804
Mean   :23.4   Mean   :5.47   Mean   :194   Mean   :104.5   Mean   :2978
3rd Qu.:29.0   3rd Qu.:8.00   3rd Qu.:276   3rd Qu.:126.0   3rd Qu.:3615
Max.   :46.6   Max.   :8.00   Max.   :455   Max.   :230.0   Max.   :5140

 acceleration       year        origin                      name
Min.   : 8.0   Min.   :70   Min.   :1.00   amc matador       :  5
1st Qu.:13.8   1st Qu.:73   1st Qu.:1.00   ford pinto        :  5
Median :15.5   Median :76   Median :1.00   toyota corolla    :  5
Mean   :15.5   Mean   :76   Mean   :1.58   amc gremlin       :  4
3rd Qu.:17.0   3rd Qu.:79   3rd Qu.:2.00   amc hornet        :  4
Max.   :24.8   Max.   :82   Max.   :3.00   chevrolet chevette:  4
                                           (Other)           :365
```

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 on the relationship of `mpg` (miles per gallon) to `horsepower`, as displayed in Figure 1 (a). 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 (a). The linear fit is clearly inappropriate; the fits for $p = 2$ (quadratic) through 4 are very similar; and the degree-5 polynomial may

[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.

over-fit the data by chasing one or two relatively high `mpg` values at the right (but see the CV results reported below).

Figure 1 (b) 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
```

which takes the following arguments:

- `model`, an `"lm"` object, the only required argument.
- `data`, the data to which the model was fit, and which can usually be inferred from the `model` object, using the `get_data()` function from the **insight** package (Lüdecke, Waggoner, and Makowski 2019). The **insight** package provides a robust way to access model components for many classes of statistical models.
- `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 Appendix A.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"` (the default) selects `"hatvalues"` for LOO CV and `"Woodbury"` for k-fold CV, both of which are much more efficient than directly recomputing the least-squares fit (see below and Section **??**).

- `ncores`, the number of cores to employ for parallel computation; if `ncores = 1` (the default), the computations are not parallelized.

To illustrate, we perform 10-fold CV for a quadratic polynomial fit to the `Auto` data:[3]

```
R> library("car", quietly = TRUE)
R>
R> m.auto <- lm(mpg ~ poly(horsepower, 2), data = Auto)
R> brief(m.auto)

           (Intercept) poly(horsepower, 2)1 poly(horsepower, 2)2
Estimate        23.446              -120.14                44.09
Std. Error       0.221                4.37                 4.37

 Residual SD = 4.37 on 389 df, R-squared = 0.688

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

R RNG seed set to 623225

cross-validation criterion (mse) = 19.409

R> summary(cv.auto)

10-Fold Cross Validation
method: Woodbury
criterion: mse
cross-validation criterion = 19.409
bias-adjusted cross-validation criterion = 19.386
95% CI for bias-adjusted CV criterion = (15.885, 22.887)
full-sample criterion = 18.985
```

The `print()` method for `"cv"` objects reports the CV estimate of MSE; the `summary()` method, which we'll use extensively in the sequel, in addition reports a bias-adjusted estimate of the MSE (the bias adjustment is explained in Appendix A.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> summary(cv(m.auto, k = "loo"))

n-Fold Cross Validation
method: hatvalues
criterion: mse
cross-validation criterion = 19.248
```

[3] We load the **car** package (Fox and Weisberg 2019) here for the `brief()` function and to use below.

The "hatvalues" method computes 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); for example:

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

n-Fold Cross Validation
method: naive
criterion: mse
cross-validation criterion = 19.248
bias-adjusted cross-validation criterion = 19.248
95% CI for bias-adjusted CV criterion = (15.779, 22.717)
full-sample criterion = 18.985
```

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; the output shown is for the CV estimate of MSE and the bias-corrected CV estimate. 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

[1] 19.248 19.248

R> set.seed(19412)
R> print(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, unit = "relative"), signif = 3)

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

Unit: relative
      expr   min    lq  mean median    uq   max neval cld
 hatvalues   1.0   1.0   1.0    1.0   1.0   1.0    10 a
  Woodbury  12.5  12.3  11.2   10.7  10.6  10.2    10 a
     naive 223.0 218.0 208.0  188.0 182.0 248.0    10  b
    cv.glm 384.0 378.0 353.0  330.0 354.0 337.0    10   c
```

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.[4]

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]

$m.2

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

Coefficients:
         (Intercept)  poly(horsepower, p)1  poly(horsepower, p)2
                23.4                 -120.1                 44.1
```

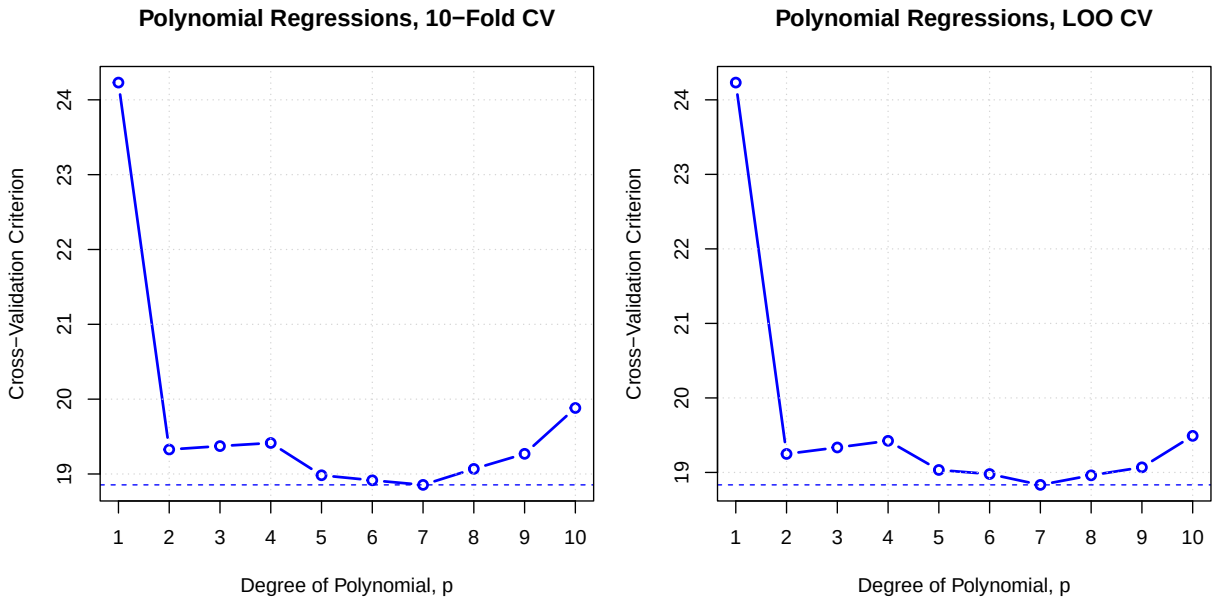
where, for example, `mlist[2]` is the quadratic fit.

We then apply `cv()` to the list of 10 models for both 10-fold and LOO CV (the `data` argument to `cv()` is required):

```
R> mlist <- models(mlist)
R>
R> cv.auto.10 <- cv(mlist, data = Auto, seed = 2120)
R> cv.auto.10[2]

Model m.2:
cross-validation criterion = 19.346
```

[4] Out of impatience, we asked `microbenchmark()` to execute each command only 10 times rather than the default 100. With the exception of the last column, the output is self-explanatory. The last column shows which methods have average timings that are statistically distinguishable (as indicated by different letters). Because of the small number of repetitions (i.e., 10), the `"hatvalues"` and `"Woodbury"` methods aren't distinguishable, but the difference between these methods persists when we perform more repetitions—we invite the patient reader to redo this computation with the default `times = 100` repetitions.



(a) (b)

Figure 2: Cross-validated (a) 10-fold and (b) LOO MSE as a function of polynomial degree, $p = 1, 2, \ldots, 10$.

```
R> cv.auto.loo <- cv(mlist, data = Auto, k = "loo")
R> cv.auto.loo[2]

Model m.2:
cross-validation criterion = 19.248
```

The `models()` function takes an arbitrary number of regression models as its arguments, which are optionally named, to create a `"modList"` object. Alternatively, as here, we can supply a pre-constructed, optionally named, list of models as the single argument to `models()`. To visualize the results, comparing the models, we invoke the `plot()` method for the `"cvModList"` objects returned by `cv()` (see Figure 2):[5]

```
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 1 (b) on page 4), except for the highest-degree polynomials, where the CV results more clearly suggest over-fitting.

[5]The objects `cv.auto.10` and `cv.auto.loo` do not include confidence intervals for the CV MSE estimates because, as we explained, there are fewer than 400 cases in the `Auto` data. Consequently, confidence intervals are not shown in the graphs in Figure 2. We can force `cv()` to compute confidence intervals by adding the argument `confint = TRUE`; we invite the reader to do this and to verify that the resulting confidence intervals are very broad, as was the case for the quadratic model fit to the `Auto` data in the preceding section.

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 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.[6]

We can think of two approaches to applying cross-validation to clustered data:[7]

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 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
```

[6] 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 crossed random effects in the **cv** package vignette on mixed models.

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

```
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 (summarizing the model with the `S()` function from the **car** package):

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


Estimates of Fixed Effects:
                   Estimate Std. Error z value Pr(>|z|)
(Intercept)          12.128      0.199   60.86  < 2e-16
mean.ses              5.333      0.369   14.45  < 2e-16
cses                  2.945      0.156   18.93  < 2e-16
sectorCatholic        1.227      0.306    4.00  6.2e-05
mean.ses:cses         1.039      0.299    3.48  0.00051
cses:sectorCatholic  -1.643      0.240   -6.85  7.3e-12

Estimates of Random Effects (Covariance Components):
 Groups   Name        Std.Dev. Corr
 school   (Intercept) 1.543
          cses        0.318    0.39
 Residual             6.060

Number of obs: 7185, groups:  school, 160

logLik     df    AIC    BIC
-23252     10  46524  46592
```

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

```
R> summary(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 = 39.157
bias-adjusted cross-validation criterion = 39.148
95% CI for bias-adjusted CV criterion = (38.066, 40.231)
full-sample criterion = 39.006
```

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

```
R> summary(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 = 37.445
bias-adjusted cross-validation criterion = 37.338
95% CI for bias-adjusted CV criterion = (36.288, 38.388)
full-sample criterion = 36.068
```

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 four artificial data sets that exemplify aspects of cross-validation particular to hierarchical models. Using these data sets, 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.

The four data sets differ in the magnitude of between-cluster variation compared to within-cluster variation. They serve to illustrate how fitting mixed models, and, consequently, the

cross-validation of mixed models, is sensitive to relative variation, which affects the degree of shrinkage of within-cluster estimates of effects towards between-cluster estimates.

The purpose of this section is to explain some of the complexities of cross-validating mixed-effects models, which motivates the approach that we take for mixed models in the **cv** package. The examples in this section introduce no features of the **cv** package that we haven't already encountered, and the generation of data for these examples involves unenlightening data management. Consequently, we don't show the code for data generation, model fitting, and graphs in this section.[8]

Consider a researcher studying the effect of the dosage x of a drug on the severity of symptoms y for a hypothetical disease. The researcher has longitudinal data on 20 patients, each of whom was observed on 5 occasions in which patients received different dosages of the drug. The data are observational, with dosages prescribed by the patients' physicians, so that patients who were more severely affected by the disease received higher dosages of the drug.

Within patients, higher dosages are generally associated with a reduction in symptoms. Between patients, however, higher dosages are associated with higher levels of symptoms. A plausible mechanism is a reversal of causality: Within patients, higher dosages alleviate symptoms, but, between patients, higher morbidity causes the prescription of higher dosages.

To generate the four data sets, we first constructed a template data set from a population with a between-patient effect of dosage of 1.0 and a within-patient effect of -0.5. We then applied four different multipliers to the between-patient standard deviation from the regression line of patient centroids. The resulting data sets, shown in Figure 3, differ in the between-patient variation of patient centroids from a common between-patient regression line, exhibiting a progression from low to high variation around the common regression line. Each patient has the same relative configuration of dosages and symptoms in each of the four data sets. To help visualize the structure of the data, estimated 50% Gaussian concentration ellipses are shown for each patient.

Using data like these, a researcher may attempt to obtain an estimate of the within-patient effect of dosage by fitting mixed models with a random intercept. We will illustrate how this approach results in estimates that are highly sensitive to the relative variation at the two levels of the mixed model by considering four models, all of which include a random intercept. The first three models are sequentially nested in their fixed effects: (1) `~ 1`, intercept only; (2) `~ 1 + x`, intercept and effect of `x`; and (3) `~ 1 + x + xm`, intercept, effect of `x`, and a contextual variable, `xm`, consisting of the within-patient mean of `x`. The final model, (4) `~ 1 + I(x - xm)`, uses an intercept and the centered-within-group variable `x - xm`. We thus fit four models to four data sets for a total of 16 models.

We proceed to obtain predictions from each model (1) using only the fixed effects, as for cross-validation based on clusters (i.e., patients), and (2) using both fixed and random effects—that is, the BLUPs—as for cross-validation based on cases (i.e., occasions within patients). The fixed- and random-effects predictions from these models are displayed in Figure 4 along with a summary of the data using 50% concentration ellipses (see the top row of the figure).

[8]The code for this section is in the complementary materials for the paper. We used the `glmmTMB()` function in the **glmmTMB** package (Brooks *et al.* 2017) for the models fit here because, in our experience, it is more likely to converge than functions in the **nlme** and the **lme4** packages for models with low between-cluster variation. We used the **lattice** (Sarkar 2008) and **latticeExtra** (Sarkar and Andrews 2022) packages for the graphs.

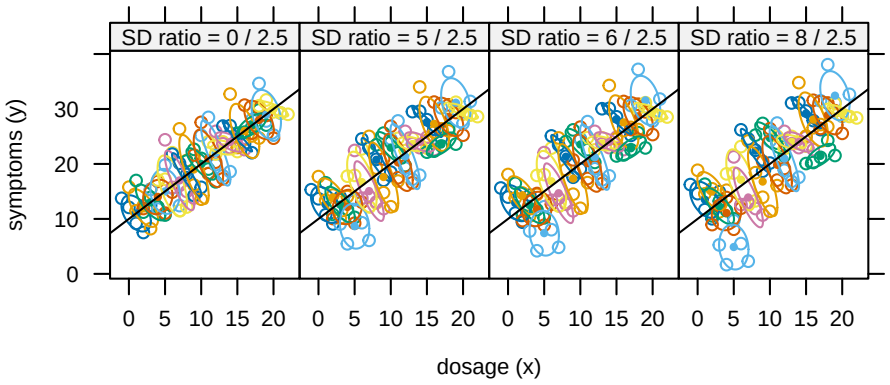


Figure 3: Data sets showing identical within-patient configurations with increasing between-patient variation. The labels above each panel show the ratio of between-patient SD to within-patient SD. The ellipses are estimated 50% Gaussian concentration ellipses for each patient. The population between-patient regression line, $\mathsf{E}(y) = 10 + x$, is shown in each panel, where y is symptoms and x is dosage.

Data sets with relatively low between-patient variation result in strong shrinkage of fixed-effects predictions, and also of BLUPs, towards the between-patient relationship between `y` and `x` in the model with an intercept and `x` as fixed-effect predictors, `~ 1 + x`. The inclusion of the contextual variable `xm` in the model corrects this problem.

Although the BLUPs fit the observed data more closely than predictions based on fixed effects alone, the slopes of within-patient BLUPs do not conform to the within-patient slopes for the `~ 1 + x` model in the two data sets with the smallest between-patient variation. For data with a small between-patient variation, fixed-effects predictions for the `~ 1 + x` model have a slope that is close to the between-patient slope but provide better overall predictions than the fixed-effect predictions for data sets with larger between-subject variation. With data whose between-patient variation is relatively large, predictions based on the model 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.

We therefore anticipate (and show later in this section) that case-based cross-validation may prefer the intercept-only model, `~ 1` to the larger `~ 1 + x` model when the between-cluster variation is relatively small, but that cluster-based cross-validation will prefer the latter to the former. We will discover that case-based cross-validation prefers the `~ 1 + x` model to the `~ 1` model for the "5 / 2.5" data set, but cluster-based cross-validation prefers the latter model to the former. The situation is entirely reversed with the "8 / 2.5" data set.

The third model, `~ 1 + x + xm`, includes a contextual effect of `x`—that is, the cluster mean `xm`—along with `x`, 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 | patient)`, which is the model that generated the data. The fit of the mixed model `~ 1 + x + xm` is consequently similar to that of a fixed-effects-only model with `x` and a categorical predictor for individual patients (i.e., `y ~ factor(patient) + x`, treating patients as a factor, and not shown here).

We next carry out case-based cross-validation, which, as we have explained, is based on both fixed and predicted random effects (i.e., BLUPs), and cluster-based cross-validation, which

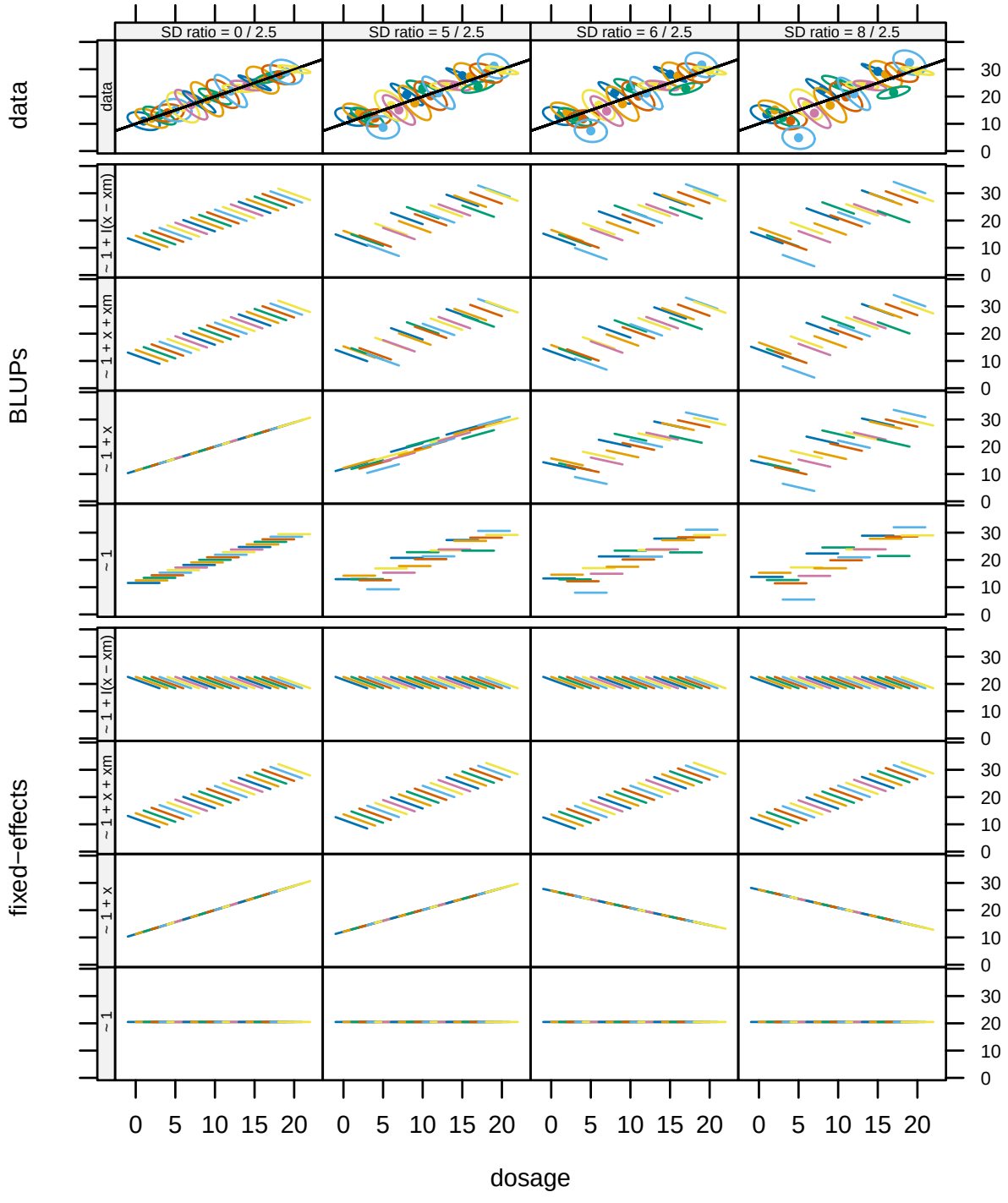


Figure 4: Fixed- and random-effect predictions (BLUPs) using each model applied to data sets with varying between- and within-patient standard-deviation ratios. The top row shows summaries of the within-patient data using estimated 50% concentration ellipses, along with the patient centroids.

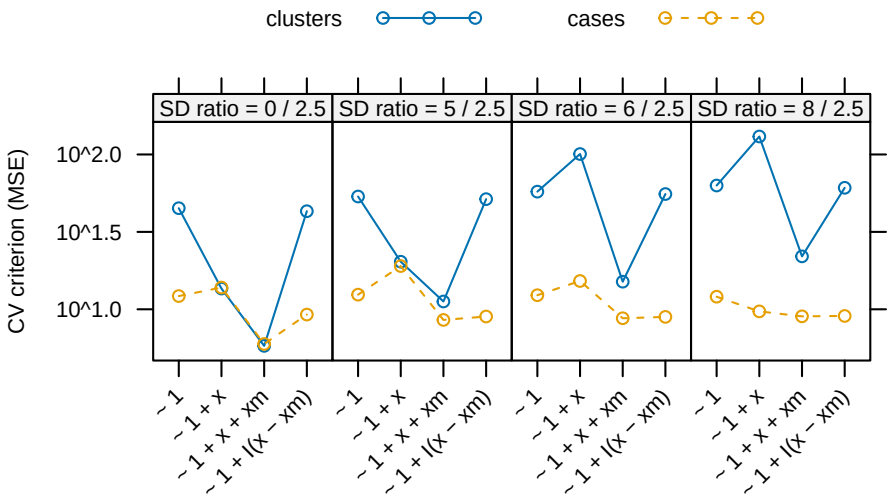


Figure 5: 10-fold cluster- and case-based cross-validation comparing mixed models with random intercepts and various fixed effects.

is based on fixed effects only.[9] The results for all data sets, using both cluster-based and case-based cross-validation, are assembled in Figure 5.

In summary, when between-cluster variation is relatively large, the model `~ 1 + x`, with `x` alone and without the contextual mean `xm` of `x`, is assessed as fitting very poorly by cluster-based CV, but relatively much better by case-based CV. In all of our examples, the model `~ 1 + x + xm`, which includes both `x` and its contextual mean, produces better results using both cluster-based and case-based CV. 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 independently 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

[9] In order to reduce between-model random variability in comparisons of models on the same data set, we applied `cv()` to a list of models created by the `models()` function (introduced previously), performing cross-validation with the same folds for each model.

Ripley 2002), performs stepwise predictor selection for regression models; `selectTrans()`, based on the `powerTransform()` function in the **car** package, transforms predictors and the response in a regression model towards normality; `selectTransStepAIC()`—the use of which we illustrate in the current section—performs both of these procedures sequentially; 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, a process that we term "meta cross-validation."[10]

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:[11]

```
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); in each panel, the solid line shows the linear least-squares fit and the broken line is for a nonparametric regression:

[10] In a vignette on extending the **cv** package, we explain how to specify additional model-selection procedures.

[11] 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.

```
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,
+                        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.[12] It's apparent that most of the numeric 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:

```
R> m.auto <- lm(mpg ~ ., data = Auto)
R> crPlots(m.auto,
+          terms = ~ displacement + horsepower + weight + acceleration,
+          pch = ".", ylab = "C+R", las = 2)
```

The component + residual ("C+R") plots in Figure 7, produced by the `crPlots()` function in the **car** package, clearly reveal the inadequacy of the model. The broken blue lines in the C+R plots show linear least-squares fits, while the solid magenta lines are nonparametric-regression smooths.

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

[12]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.

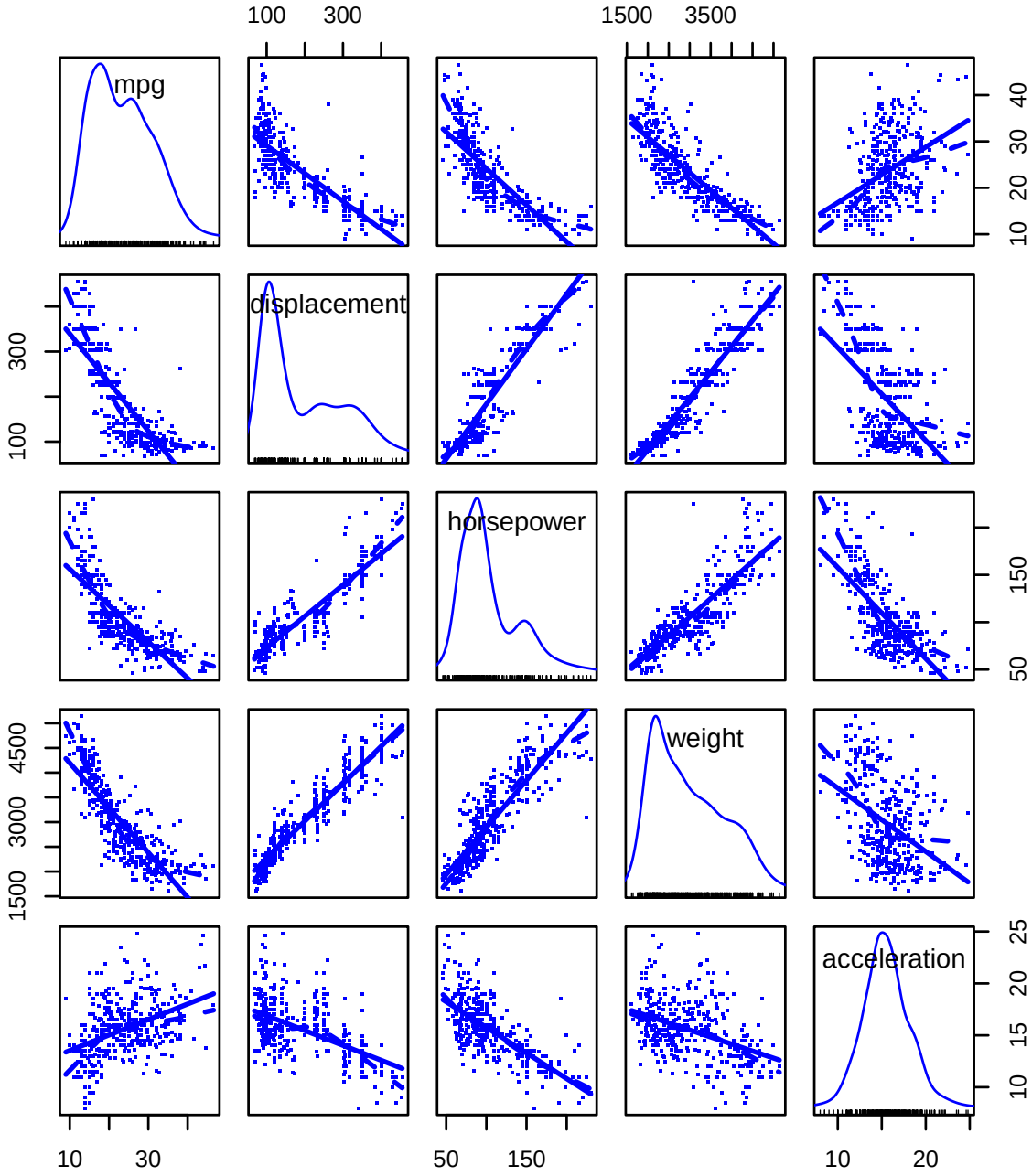


Figure 6: Scatterplot matrix for the numeric variables in the `Auto` data.

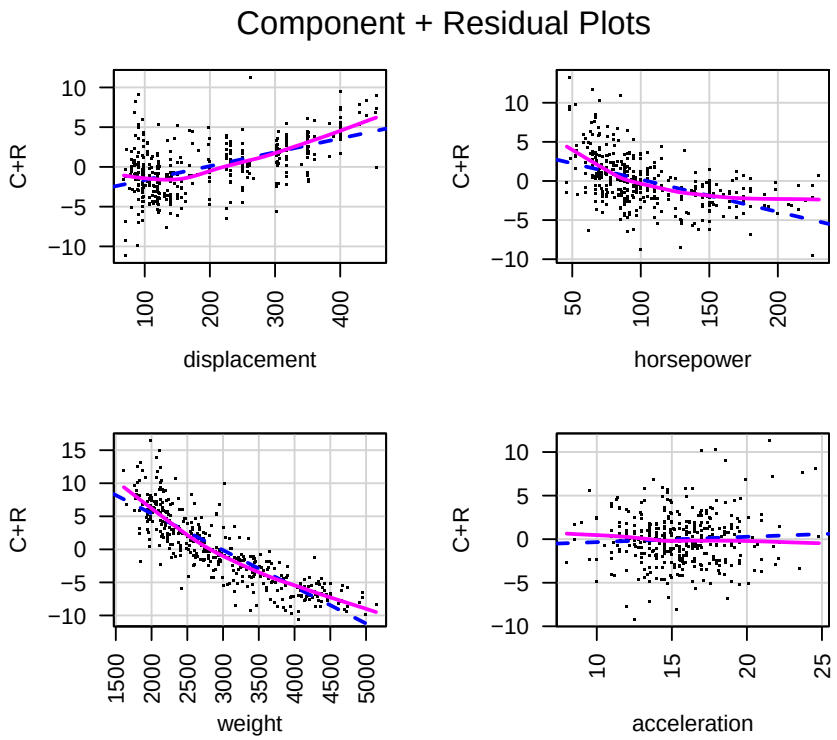


Figure 7: Component + residual plots for the numeric predictors in the working model fit to the `Auto` data.

Cox 1964). Although the multinormal likelihood is maximized, the variables are transformed individually. 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 zeros: 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) 4.8729  4 0.301

Likelihood ratio test that no transformations are needed
                                LRT df   pval
LR test, lambda = (1 1 1 1) 390.08  4 <2e-16
```

We then apply the (rounded) transformations—all, as it turns out, logs (i.e., "zeroth" 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) 0.00080154  1 0.977

Likelihood ratio test that no transformation is needed
                          LRT df   pval
LR test, lambda = (1) 124.13  1 <2e-16

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

The transformed numeric variables are much better-behaved and the partial relationships in the model fit to the transformed data are much more nearly linear (as we invite the reader to verify by redrawing the scatterplot matrix for the transformed variables and the C+R plots for the regression fit to the transformed data).

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> brief(m.step)

           (Intercept) horsepower weight acceleration year71   year72  year73
Estimate         9.435    -0.2763 -0.609      -0.1314 0.0280 -0.00711 -0.0395
Std. Error       0.262     0.0561  0.056       0.0532 0.0289  0.02845  0.0260
           year74 year75 year76 year77 year78 year79 year80 year81 year82
Estimate   0.0528 0.0532 0.0743 0.1379 0.1459 0.2360 0.3353 0.2629 0.3234
Std. Error 0.0300 0.0293 0.0282 0.0289 0.0275 0.0291 0.0311 0.0306 0.0296
           originEurope originJapan
Estimate         0.0558      0.0436
Std. Error       0.0168      0.0175

 Residual SD = 0.105 on 374 df, R-squared = 0.909
```

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] 6.5121
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] 8.0932
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.[13]

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

[13]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 Appendix A.2). Yet another possibility, suggested to us by a reviewer of an earlier version of this paper, is to compute the MSE for both models on the log-response scale rather than on the original scale of the response. This suggestion is quite general, in that the essential point is to use the *same* response scale for all models to be compared, not necessarilty the *original* scale of the response.

```
R> num.predictors

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

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> summary(cvs)

10-Fold Cross Validation
criterion: mse
cross-validation criterion = 7.4856
bias-adjusted cross-validation criterion = 7.3435
full-sample criterion = 6.5121
```

The other arguments to `cv()` are:[14]

- `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(cvs)`, which isn't shown), we see that there is little uncertainty in choosing variable transformations, 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`.

[14] See `?cv.function` for additional optional arguments and details.

4.2. Example: Applying meta CV to polynomial regression for the Auto data[15]

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 meta-CV. 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:[16]

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

R RNG seed set to 2120

R> summary(metaCV.auto)

10-Fold Cross Validation
criterion: mse
cross-validation criterion = 20.012
bias-adjusted cross-validation criterion = 20.619
full-sample criterion = 18.746

R> brief(m.sel <- cvInfo(metaCV.auto, "selected model"))

           (Intercept) poly(horsepower, p)1 poly(horsepower, p)2
Estimate        23.446              -120.1                44.1
Std. Error       0.217                 4.3                 4.3
           poly(horsepower, p)3 poly(horsepower, p)4 poly(horsepower, p)5
Estimate                 -3.95                -5.19                13.3
Std. Error                4.30                 4.30                 4.3
           poly(horsepower, p)6 poly(horsepower, p)7
Estimate                 -8.55                 7.98
```

[15] What we call here "meta 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).

[16] Two functions are employed to access information in objects produced by functions in the **cv** package: `cvInfo()` and `as.data.frame()` are generic functions (the latter a standard-R generic) with methods for classes `"cv"`, `"cvList"`, `"cvModlist"`, and `"cvSelect"` (in the case of `cvInfo()`) , and `"cv"`, `"cvList"`, and `"cvModlist"` (in the case of `as.data.frame()`). Here we use `cvInfo(metaCV.auto, "selected model")` to retrieve the model selected by `cv()` via `selectModelList()`. See `?cv` and `?cvInfo.cvSelect` for details.

```
Std. Error                  4.30                  4.30

 Residual SD = 4.3 on 384 df, R-squared = 0.702
```

```
R> cv(m.sel, seed = 2120)

R RNG seed set to 2120

cross-validation criterion (mse) = 18.898
```

As expected, meta CV produces a larger estimate of MSE for the selected 7th degree polynomial model than CV applied directly to this model (using the same seed for R's random-number generator to obtain the same folds).

We can equivalently call `cv()` with the list of models as its first argument and set the argument `meta = TRUE` (identical output not shown):

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

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 (including of (3)) 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 UK 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.   :24.0   Min.   :1.00
 Labour          :720   1st Qu.:41.0   1st Qu.:3.00
 Liberal Democrat:343   Median :53.0   Median :3.00
                        Mean   :54.2   Mean   :3.25
                        3rd Qu.:67.0   3rd Qu.:4.00
                        Max.   :93.0   Max.   :5.00
 economic.cond.household     Blair          Hague         Kennedy
```

```
Min.   :1.00           Min.   :1.00   Min.   :1.00   Min.   :1.00
1st Qu.:3.00           1st Qu.:2.00   1st Qu.:2.00   1st Qu.:2.00
Median :3.00           Median :4.00   Median :2.00   Median :3.00
Mean   :3.14           Mean   :3.33   Mean   :2.75   Mean   :3.14
3rd Qu.:4.00           3rd Qu.:4.00   3rd Qu.:4.00   3rd Qu.:4.00
Max.   :5.00           Max.   :5.00   Max.   :5.00   Max.   :5.00
    Europe      political.knowledge    gender
Min.   : 1.00   Min.   :0.00        female:812
1st Qu.: 4.00   1st Qu.:0.00        male  :713
Median : 6.00   Median :2.00
Mean   : 6.73   Mean   :1.54
3rd Qu.:10.00   3rd Qu.:2.00
Max.   :11.00   Max.   :3.00
```

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, trace = FALSE)
```

Figure 8 shows an "effect plot," using the the **effects** package (Fox and Weisberg 2019) to visualize the `Europe` × `political.knowledge` 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")),
+      axes = list(x = list(rug = FALSE), y = list(style = "stacked")))
```

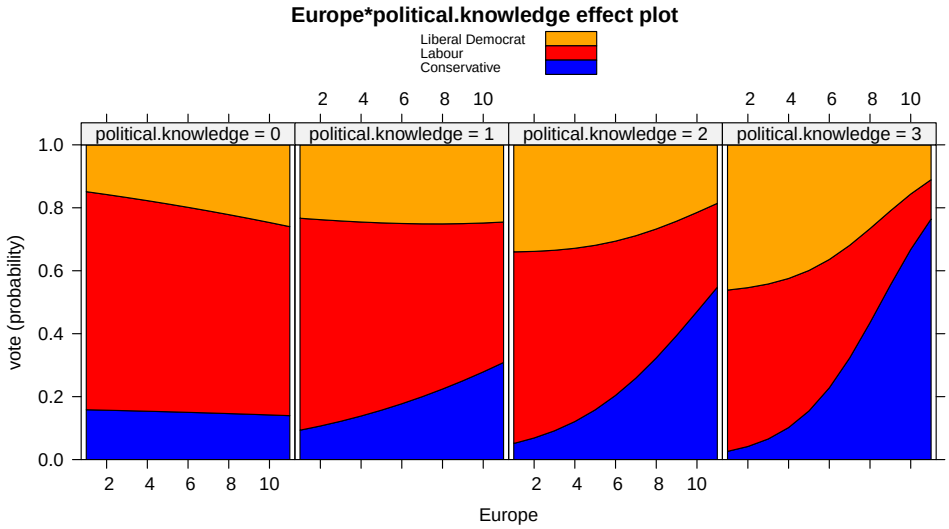


Figure 8: Effect plot for the interaction between attitude towards European integration and political knowledge in the multinomial logit model fit to voting data from the 2001 British Election Panel Study, using party colors.

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)
    attr(result, "casewise loss") <- "y != round(yhat)"
    result
}
<bytecode: 0x11f77aed0>
<environment: namespace:cv>
```

After doing some error checking, `BayesRule()` rounds the predicted probability `yhat` 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 Appendix A.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] 0.31869
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. The argument `yhat` to our `BayesRuleMulti()` function is the vector of Bayes's rule categorical predictions, while `y` is the vector of observed categorical responses. The function calculates and returns 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
         0.30295          0.47213          0.22492
```

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,

```
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

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

R RNG seed set to 3465

10-Fold Cross Validation
criterion: BayesRuleMulti
cross-validation criterion = 0.32459
bias-adjusted cross-validation criterion = 0.32368
95% CI for bias-adjusted CV criterion = (0.30017, 0.34718)
full-sample criterion = 0.31869
```

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 software for cross-validation

In this section with briefly compare the `cv` package to other packages in R that support cross-validation, and then even more briefly consider facilities for cross-validation in other commonly used statistical software, SAS, Stata, and Python.

6.1. Other R software

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,[17] 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

[17] R users with sophisticated programming skills would generally find it unchallenging to implement CV directly for specific applications (as we illustrate below). 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.

(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 implements 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), comparing LOO CV computed using the `cv()` function in the **cv** package to LOO CV computed using the **caret** and **rsample** packages. At more than 15 million cumulative downloads, **caret** is by a very wide margin the most downloaded from the RStudio CRAN mirror of the R packages mentioned above. Among the packages that focus more on cross-validation, **rsample** is the most downloaded, at nearly 4 million downloads.

Here, with minimal explanation, is how one would perform LOO CV for this example using **rsample**:[18]

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

[1] 1470

R> mod_form <- Attrition ~ JobSatisfaction + Gender + MonthlyIncome
R>
R> rs_obj <- loo_cv(attrition)
R>
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
```

[18] 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.

```
+   res
+ }
R>
R> library("purrr", warn.conflicts=FALSE)
R>
R> rs_obj$results <- map(rs_obj$splits, holdout_results, mod_form)
R> rs_obj$accuracy <- map_dbl(rs_obj$results, function(x) mean(x$correct))
R> 1 - summary(rs_obj$accuracy)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   0.000   0.000   0.161   0.000   0.000
```

The response variable in this example, `Attrition`, is binary. The function `holdout_results()` must be supplied by the user, and mean "`accuracy`" is the complement of the Bayes-rule classification error rate.

In this case, it's simple (and, in our opinion, simpler) to program LOO CV directly. For example,

```
R> mod.attrition <- glm(mod_form, data = attrition, family = binomial)
R> n <- nrow(attrition)
R> yhat <- numeric(n)
R> for (i in 1:n){
+   m <- update(mod.attrition, data = attrition[-i, ])
+   yhat[i] <- predict(m, newdata = attrition[i, ], type = "response")
+ }
R> mean(((attrition$Attrition == "Yes") - round(yhat)) ^ 2)

[1] 0.16122
```

The **caret** package supports generalized linear models fit by `glm()` and can straightforwardly perform LOO CV:

```
R> library("ggplot2", warn.conflicts = FALSE)
R> library("caret", quietly = TRUE, warn.conflicts = FALSE)
R> train(x = attrition[, c("JobSatisfaction", "Gender", "MonthlyIncome")],
+       y = attrition$Attrition, method = "glm",
+       trControl = trainControl(method = "LOOCV"))

Generalized Linear Model

1470 samples
   3 predictor
   2 classes: 'No', 'Yes'

No pre-processing
Resampling: Leave-One-Out Cross-Validation
```

```
Summary of sample sizes: 1469, 1469, 1469, 1469, 1469, 1469, ...
Resampling results:

  Accuracy  Kappa
  0.83878   0
```

where "`Accuracy`," again, is the complement of the misclassification rate.

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

```
R> cv(mod.attrition, k = "loo", criterion = BayesRule)

cross-validation criterion (BayesRule) = 0.16122

R> 1 - mean(rs_obj$accuracy)

[1] 0.16122
```

Let's compare the computational efficiency of the various 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"`:[19]

```
R> set.seed(53437)
R> print(microbenchmark::microbenchmark(
+   cv.hatvalues = cv(mod.attrition, k = "loo", criterion = BayesRule,
+                     method = "hatvalues"),
+   cv.wood = cv(mod.attrition, k = "loo", criterion = BayesRule,
+                method = "Woodbury"),
+   cv.exact = cv(mod.attrition, k = "loo", criterion = BayesRule),
+   direct = {
+     n <- nrow(attrition)
+     yhat <- numeric(n)
+     for (i in 1:n){
+       m <- update(mod.attrition, data = attrition[-i, ])
+       yhat[i] <- predict(m, newdata = attrition[i, ], type = "response")
+     }
+     mean(((attrition$Attrition == "Yes") - round(yhat)) ^ 2)
+   },
+   rsample = {
+     rs_obj$results <- map(rs_obj$splits, holdout_results, mod_form)
+     rs_obj$accuracy <- map_dbl(rs_obj$results,
+                                function(x) mean(x$correct))
+   },
+   caret = train(x = attrition[, c("JobSatisfaction", "Gender",
```

[19] As before, we ask `microbenchmark()` to report relative timings, and also as before, we set `times = 10` rather than the default `times = 100`.

```
+                                    "MonthlyIncome")],
+               y = attrition$Attrition,
+               method = "glm",
+               trControl = trainControl(method = "LOOCV")),
+   times = 10, unit = "relative"), signif = 3)

Unit: relative
         expr    min     lq   mean median     uq    max neval    cld
 cv.hatvalues    1.0    1.0    1.0    1.0    1.0    1.0    10 a
      cv.wood   83.1   71.7   71.8   70.3   69.4   65.8    10  b
     cv.exact 3790.0 3150.0 3140.0 3030.0 2990.0 2930.0    10   c
       direct 4230.0 3530.0 3520.0 3450.0 3370.0 3170.0    10    d
      rsample 4680.0 3910.0 3870.0 3770.0 3700.0 3460.0    10     e
        caret 5550.0 4610.0 4560.0 4450.0 4330.0 4090.0    10      f
```

Computing time for **rsample** and **caret** for this problem is similar to direct computation of LOO CV and to `cv()` with `method="exact"`, but almost two orders of magnitude slower than `cv()` with `method="Woodbury"`, and more than three orders of magnitude slower than `cv()` with `method="hatvalues"`.

6.2. Comparing cv to other software for cross-validation: SAS, Stata, and Python

Although we do not have recent experience with statistical software for cross-validation other than in R,[20] we have examined provisions for CV in SAS, Stata, and Python, primarily by locating and reading documentation.

SAS

Cross-validation in SAS is associated with particular procedures ("PROCs"), and is distributed across several products, including, for example, `PROC LOGISTIC` and `PROC PLS` in SAS/STAT (SAS Institute 2020b), in SAS Enterprise Miner (SAS Institute 2018), and in SAS Visual Data Mining and Machine Learning (SAS Institute 2020a). These appear to be straightforward implementations of k-fold CV.

In addition, SAS has programming capabilities, including via its macro language, and we're aware of several independent implementations of CV in SAS macros. These tend to be limited in scope. For example Slaets, Schmitter, Hilger, Lamers, Piepho, Vien, and Cadisch (2014) employ and provide access to a SAS macro for k-fold CV of mixed-effects models, which repeatedly calls `PROC MIXED` in SAS/STAT, and which performs cluster-based resampling (one of the options that we provide for mixed-effects models in the **cv** package).

Stata

[20] It is our impression that these three statistical systems, in addition to R, are most likely to be used for applications of cross-validation. We were once frequent users of SAS, though not recently, and have only cursory experience with Stata and Python, so we are not ideally situated to evaluate CV software for any of the three.

As far as we can tell, Stata has no standard implementation of cross-validation. Like R, however, Stata is extensible through a programming language that allows users to define new commands in so-called "ado" files.

Several such contributed Stata CV commands, varying in generality, are shared in an archive maintained at the Boston College Department of Economics. For example, the `CV_REGRESS` command (Rios-Avila 2018) computes leave-one-out CV for least-squares regression using the hatvalues, and thus is comparable to `cv()` for linear models with `method = "hatvalues"`. In contrast, the `CROSSVALIDATE` command (Schonlau 2020) performs k-fold CV for a wide range of regression models fit in Stata, and is roughly comparable to the default method of `cv()`, although `CROSSVALIDATE` returns cross-validated fitted values, from which cost criteria can be computed, rather than (as `cv()`) cost-criteria directly.

Python

The programming language Python is often the software of choice for "machine learning," in which cross-validation can play a prominent role. Machine learning and statistical modeling more generally are supported through extensive Python libraries, such as **scikit-learn** (Pedregosa, Varoquaux, Gramfort, Michel, Thirion, Grisel, Blondel, Prettenhofer, Weiss, Dubourg, Vanderplas, Passos, Cournapeau, Brucher, Perrot, and Duchesnay 2011; Scikit-Learn Documentation Team 2024) and **TensorFlow** (Abadi, Agarwal, Barham, Brevdo, Chen, Citro, Corrado, Davis, Dean, Devin, Ghemawat, Goodfellow, Harp, Irving, Isard, Jia, Jozefowicz, Kaiser, Kudlur, Levenberg, Mané, Monga, Moore, Murray, Olah, Schuster, Shlens, Steiner, Sutskever, Talwar, Tucker, Vanhoucke, Vasudevan, Viégas, Vinyals, Warden, Wattenberg, Wicke, Yu, and Zheng 2015, which isn't limited to use within Python).

Functions for cross-validation are provided in **scikit-learn**, which supports k-fold, repeated k-fold, and leave-one-out CV. These functions can be applied to data sets consisting of independent observations, hierarchical groups, or time-series data. A sample program in the **scikit-learn** documentation (Scikit-Learn Documentation Team 2024) demonstrates how to implement meta cross-validation (termed "nested cross-validation" in the documentation).

The approach taken by **scikit-learn** for independent observations is similar to that in the default `cv()` method in the **cv** package. The approach for hierarchical data is similar, but not identical, to that in **cv** (as described in Section 3). Our **cv** package doesn't currently provide for cross-validating time-series models, but we anticipate that the next version of the package will.

Appendix: Computational notes

A.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 is 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) [21]

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

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 $h_i = \mathbf{x}_i^\top\left(\mathbf{X}^\top\mathbf{W}\mathbf{X}\right)^{-1}\mathbf{x}_i$ (the so-called "hatvalue"). Here, $\mathbf{x}_i^\top$ is the ith row of $\mathbf{X}$, and $\mathbf{x}_i$ 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. Then 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 identity (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-m matrix. We apply this result by letting

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

where the subscript $\mathbf{j} = (i_{j1}, \ldots, i_{jm})^\top$ represents the vector of indices for the $m = n_j \approx n/k$ cases in the jth fold. 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 $\mathbf{A} = \mathbf{X}^\top\mathbf{W}\mathbf{X}$ 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.)

[21] 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 QR decomposition of the model matrix with pivoting to obtain computationally more stable results.

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.

A.2 Computation of the bias-corrected CV criterion and confidence intervals

Let $\text{CV}(\mathbf{y}, \widehat{\mathbf{y}})$ 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 $\text{CV}(\mathbf{y}, \widehat{\mathbf{y}})$ is the mean of casewise components, that is, $\text{CV}(\mathbf{y}, \widehat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n} \text{cv}(y_i, \widehat{y}_i)$.[22] For example, $\text{MSE}(\mathbf{y}, \widehat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2$.[23]

We divide the n cases into k folds of approximately $n_j \approx n/k$ cases each, $j = 1, \ldots, k$, where $n = \sum n_j$. As above, let $\mathbf{j}$ denote the indices of the cases in the jth fold.

Now define $\text{CV}_j = \text{CV}(\mathbf{y}, \widehat{\mathbf{y}}^{(j)})$. 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 $\text{CV} = \text{CV}(\mathbf{y}, \widehat{\mathbf{y}}^{(-i)})$. Following Davison and Hinkley (1997, 293–295), the bias-adjusted cross-validation criterion is

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

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

[23] 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.

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

Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow I, Harp A, Irving G, Isard M, Jia Y, Jozefowicz R, Kaiser L, Kudlur M, Levenberg J, Mané D, Monga R, Moore S, Murray D, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker P, Vanhoucke V, Vasudevan V, Viégas F, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y, Zheng X (2015). "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems." Software available from tensorflow.org, URL https://www.tensorflow.org/.

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.

Barnard GA (1974). "Discussion of Professor Stone's paper." *Journal of the Royal Statistical Society B*, **36**, 133–135.

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 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*. 3rd edition. Sage, Thousand Oaks CA.

Fox J, Weisberg S (2019). *An R Companion to Applied Regression*. 3rd 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*. 2nd edition. Springer-Verlag, New York.

Hastie T, Tibshirani R, Friedman J (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd edition. Springer-Verlag, 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*. 2nd edition. Springer-Verlag, 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.

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011). "Scikit-Learn: Machine Learning in Python." *Journal of Machine Learning Research*, **12**(85), 2825–2830. ISSN 1533-7928. URL http://jmlr.org/papers/v12/pedregosa11a.html.

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-Verlag, New York.

Raudenbush SW, Bryk AS (2002). *Hierarchical Linear Models: Applications and Data Analysis Methods*. 2nd edition. Sage, Thousand Oaks CA.

Rios-Avila F (2018). "CV_REGRESS: Stata module to estimate the leave-one-out error for linear regression models." Statistical Software Components, Boston College Department of Economics. URL https://ideas.repec.org/c/boc/bocode/s458469.html.

Sarkar D (2008). *Lattice: Multivariate Data Visualization with R*. Springer-Verlag, New York. ISBN 978-0-387-75968-5. URL http://lmdvr.r-forge.r-project.org.

Sarkar D, Andrews F (2022). ***latticeExtra**: Extra Graphical Utilities Based on **lattice***. R package version 0.6-30, URL https://CRAN.R-project.org/package=latticeExtra.

SAS Institute (2018). *SAS® Enterprise Miner™ 15.3: Reference Help*. SAS Institute, Cary NC. URL https://documentation.sas.com/api/docsets/emgsj/15.2/content/emgsj.pdf.

SAS Institute (2020a). *SAS® Visual Data Mining and Machine Learning: Procedures*. SAS Institute, Cary NC. URL https://documentation.sas.com/api/collections/pgmsascdc/v_006/docsets/casml/content/casml.pdf?locale=en#nameddest=titlepage.

SAS Institute (2020b). *SAS/STAT® 15.2 User's Guide*. SAS Institute, Cary NC. URL https://documentation.sas.com/api/docsets/statug/15.2/content/statug.pdf?locale=en#nameddest=titlepage.

Schonlau M (2020). "CROSSVALIDATE: Stata module to perform k-fold crossvalidation." Statistical Software Components, Boston College Department of Economics. URL https://ideas.repec.org/c/boc/bocode/s458860.html.

Scikit-Learn Documentation Team (2024). "Nested versus Non-Nested Cross-Validation." URL https://scikit-learn.org/stable/auto_examples/model_selection/plot_nested_cross_validation_iris.html.

Slaets JIF, Schmitter P, Hilger T, Lamers M, Piepho HP, Vien TD, Cadisch G (2014). "A turbidity-based method to continuously monitor sediment, carbon and nitrogen flows in mountainous watersheds." *Journal of Hydrology*, **513**, 45–57.

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*. 4th edition. Springer-Verlag, New York.

Weisberg S (2014). *Applied Linear Regression*. 2nd edition. John Wiley & Sons, 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
E-mail:
URL: http://blackwell.math.yorku.ca/gmonette

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