Functionals in the modeling workflow

A model-building pipeline is a sequence of loops: fit one model per group, one per resample, one per hyperparameter setting, one per bootstrap replicate. Each loop is embarrassingly parallel, each benefits from a progress bar on a slow run, and each should be reproducible and should not collapse because one fit failed. That is exactly the surface `functionals` covers, with the same handful of arguments throughout.

Fit one model per group

`fmapg()` splits a data frame by one or more columns and hands each piece to `.f`.

```
by_species <- fmapg(iris, function(d) {
  fit <- lm(Sepal.Length ~ Petal.Length, data = d)
  c(slope = unname(coef(fit)[2]), r2 = summary(fit)$r.squared)
}, by = "Species")

do.call(rbind, by_species)
#>                slope         r2
#> setosa     0.5422926 0.07138289
#> versicolor 0.8282810 0.56858983
#> virginica  0.9957386 0.74688439
```

Fit many model specifications

Row-wise over a table of formulas with `fmapr()`, where each row arrives as a named list:

```
specs <- data.frame(
  response  = "mpg",
  predictor = c("wt", "hp", "wt + hp", "wt * hp"),
  stringsAsFactors = FALSE
)

aic_tbl <- fmapr(specs, function(row) {
  f   <- reformulate(row$predictor, row$response)
  fit <- lm(f, data = mtcars)
  data.frame(model = deparse(f), aic = AIC(fit), adj_r2 = summary(fit)$adj.r.squared)
})
do.call(rbind, aic_tbl)
#>           model      aic    adj_r2
#> 1      mpg ~ wt 166.0294 0.7445939
#> 2      mpg ~ hp 181.2386 0.5891853
#> 3 mpg ~ wt + hp 156.6523 0.8148396
#> 4 mpg ~ wt * hp 145.6109 0.8724170
```

Or over a grid of tuning values with `fmapn()`, iterating aligned lists in lockstep:

```
grid <- expand.grid(df = c(2, 3, 4, 5), span = NA)
fits <- fmapn(list(df = grid$df), function(df) {
  fit <- smooth.spline(mtcars$wt, mtcars$mpg, df = df)
  c(df = df, gcv = fit$cv.crit)
```

```
})
do.call(rbind, fits)
#>      df      gcv
#> [1,]  2 9.892850
#> [2,]  3 7.905117
#> [3,]  4 8.111913
#> [4,]  5 8.340277
```

Cross-validation

`fcv()` is `fapply()` named for the job: map an evaluation function over a list of resample splits. Collect the per-fold rows and `rbind()` once.

```
set.seed(42)
folds <- rsample::vfold_cv(mtcars, v = 5)

cv_rows <- fcv(folds$splits, function(split) {
  tr <- rsample::analysis(split)
  te <- rsample::assessment(split)
  fit  <- lm(mpg ~ wt + hp, data = tr)
  pred <- predict(fit, newdata = te)
  data.frame(rmse = sqrt(mean((te$mpg - pred)^2)))
})

cv <- do.call(rbind, cv_rows)
c(mean_rmse = mean(cv$rmse), sd_rmse = sd(cv$rmse))
#> mean_rmse   sd_rmse
#> 2.7863370 0.3928442
```

For a slow learner, add `ncores` and `pb = TRUE` and nothing else changes:

```
cv_rows <- fcv(folds$splits, evaluate_one, ncores = 5, pb = TRUE)
```

If a fold can fail (a degenerate resample, a non-converging fit), use `.on_error = "pass"` and drop the failures afterwards:

```
safe_rows <- fcv(folds$splits, function(split) {
  tr <- rsample::analysis(split)
  if (nrow(tr) < 3) stop("degenerate fold")
  data.frame(rmse = sd(rsample::assessment(split)$mpg))
}, .on_error = "pass")

ok <- !vapply(safe_rows, inherits, logical(1), "functionals_error")
do.call(rbind, safe_rows[ok])
#>       rmse
#> 1 7.019191
#> 2 6.837084
#> 3 4.124076
#> 4 6.542680
#> 5 5.123183
```

Bootstrap with reproducible streams

`frepeat()` evaluates an expression `times` times. With `.seed`, replicate `i` draws from its own RNG stream, so the estimate is identical on a re-run and identical whether you use one core or eight.

```r
boot_coef <- frepeat(times = 200, .seed = 2024, simplify = TRUE, expr = function() {
  idx <- sample(nrow(mtcars), replace = TRUE)
  coef(lm(mpg ~ wt, data = mtcars[idx, ]))[["wt"]]
})

c(
  estimate = coef(lm(mpg ~ wt, data = mtcars))[["wt"]],
  boot_se  = sd(boot_coef),
  ci_lo    = unname(quantile(boot_coef, 0.025)),
  ci_hi    = unname(quantile(boot_coef, 0.975))
)
#>   estimate    boot_se      ci_lo      ci_hi
#> -5.3444716  0.7310559 -6.6952511 -4.0490296
```

```r
# same result, more cores; .seed is invariant to ncores
b2 <- frepeat(times = 200, .seed = 2024, simplify = TRUE, ncores = 2, expr = function() {
  idx <- sample(nrow(mtcars), replace = TRUE)
  coef(lm(mpg ~ wt, data = mtcars[idx, ]))[["wt"]]
})
identical(round(sd(boot_coef), 10), round(sd(b2), 10))
#> [1] TRUE
```

A benchmark over learners

Put the pieces together: a list of fit/predict pairs, cross-validated on the same folds, robust to a learner that errors, reproducible via `.seed`.

```r
learners <- list(
  lm_wt      = function(tr) lm(mpg ~ wt, data = tr),
  lm_wt_hp   = function(tr) lm(mpg ~ wt + hp, data = tr),
  lm_full    = function(tr) lm(mpg ~ ., data = tr)
)

cv_rmse <- function(fit_fun) {
  rows <- fcv(folds$splits, function(split) {
    tr <- rsample::analysis(split); te <- rsample::assessment(split)
    p  <- predict(fit_fun(tr), newdata = te)
    sqrt(mean((te$mpg - p)^2))
  })
  mean(unlist(rows))
}

bench_tbl <- fmap(learners, cv_rmse, .on_error = "fill", .fill = NA_real_)
data.frame(learner = names(bench_tbl), cv_rmse = round(unlist(bench_tbl), 3))
#>           learner cv_rmse
#> lm_wt       lm_wt   3.148
#> lm_wt_hp lm_wt_hp   2.786
#> lm_full   lm_full   3.497
```

Swap `fmap(learners, cv_rmse, ...)` for `fmap(learners, cv_rmse, ncores = 3, pb = TRUE)` to run the learners in parallel with a progress bar; the rest of the code is unchanged.

Note on non-forkable models

`torch`, `xgboost`, `keras`, database handles and similar hold external-pointer state that does not survive a fork. Build such objects *inside* `.f` (so each worker makes its own) rather than capturing them from the caller. When that is not possible, keep that loop sequential (`ncores = NULL`) and parallelise a different level: the folds, the learners, or the bootstrap.