Cross-Validation

Andrew Zieffler
November 06, 2020

In the previous set of notes, we examined predictors of average life expectancy in a state, using data from *states-2019.csv*. We used several model building strategies and performance metrics to select and evaluate predictors. Unfortunately, we were evaluating the model/predictors using the same data that we used to build the model. This has several problems in practice, especially if you are using inferential metrics (*p*-values) to select the predictors:

- **Increased Type I error rates:** If you use *p*-values to select predictors, there are many tests you are examining across the model selection process. You probably need to adjust the *p*-values obtained from these tests to accommodate this.
- **Overfit/Lack of generalization:** The adopted model still has the potential to not perform well on future data.

Preparation

```
# Load libraries
library(car)
library(corrr)
library(modelr)
library(patchwork)
library(tidyverse)
library(tidymodels) # Loads broom, rsample, parsnip, recipes, workflow,
        tune, yardstick, and dials

# Import and view data
usa = read_csv("../data/states-2019.csv")
head(usa)
```

```
## # A tibble: 6 x 9
##   state  life_expectancy population income illiteracy murder hs_grad
frost  area
##   <chr>            <dbl>      <dbl>  <dbl>      <dbl>  <dbl>   <dbl>
<dbl> <dbl>
## 1 Alaba…            75.4       4.90    23.6       14.8    7.8    85.3
42.8  5.08
## 2 Alaska            78.8       0.732   33.1        9.2    8.2    92.4 200.
57.1
## 3 Arizo…            79.9       7.28    25.7       13.1    5.1    86.5
90.8 11.4
## 4 Arkan…            75.9       3.02    22.9       13.7    7.2    85.6
62.5  5.21
## 5 Calif…            81.6      39.5     30.4       23.1    4.4    82.5
27.5 15.6
## 6 Color…            80.5       5.76    32.4        9.9    3.7    91.1 168.
10.4
```

```
# Create standardized variables after removing state names
z_usa = scale(usa[ , -1]) %>%
  data.frame()
```

From our previous notes, there were a several candidate models that we may have been considering based on their AICc values. In this set of notes we will consider the best *k*-predictor models that had an AICc value within four of the minimum AICc value. These models include the following predictors:

- income
- income, population
- income, population, illiteracy rate
- income, population, illiteracy rate, murder rate

Evaluating Predictors: Cross-Validation

A better method for model selection, that does not use testing, is *cross-validation*. To perform cross-validation, you randomly split the data set into two parts: a *training* set, and a *validation* set. Any candidate models are then fitted on the training data and evaluated using the validation data. Because we use a different data set to evaluate the models, the resulting evaluation is not biased toward the data we used to fit the model (i.e., less overfit). Because of this, we get better indications about the generalizability of the models to new/future data.

In our regression example, the algorithm for performing the cross-validation is:

1. Randomly divide the `z_usa` data into two sets of observations; training and validation data sets.
2. Fit the four candidate models to the training observations.
3. Use the estimated coefficients from those fitted models to estimate the fitted values and residuals for the observations in the validation data.
4. Based on the residuals from the validation observations, compute measures of model performance (e.g., MSE).

Next, we will explore and carry out each of the steps in this algorithm to adopt a model for the `z_usa` data.

Divide the Data into a Training and Validation Set

There are many ways to do this, but I will use the `sample()` function to randomly sample 35 case numbers (about 2/3 of the data), from 1 to 52 (the number of rows in the `z_usa` data), to make up the training data. Then I use the `filter()` function to select those cases. The remaining cases (those 17 observations not sampled) are put into a validation set.

```
# Make the random sampling replicable
set.seed(42)

# Select the cases to be in the training set
training_cases = sample(1:nrow(z_usa), size = 35, replace = FALSE)

# Create training data
train = z_usa %>%
  filter(row_number() %in% training_cases)

# Create validation data
validate = z_usa %>%
  filter(!row_number() %in% training_cases)
```

Fit the Candidate Models to the Training Data

We now fit the four candidate models to the observations in the training data.

```
lm.1 = lm(life_expectancy ~ -1 + income,
        data = train)
lm.2 = lm(life_expectancy ~ -1 + income + population,
        data = train)
lm.3 = lm(life_expectancy ~ -1 + income + population + illiteracy,
        data = train)
lm.4 = lm(life_expectancy ~ -1 + income + population + illiteracy + murder,
        data = train)
```

Fit the Models Obtained to the Validation Observations

Now we can obtain the predicted life expectancies for the observations in the validation data based on the coefficients from the models fitted to the training data.

```
# Get the predicted values for the validation data
yhat_1 = predict(lm.1, newdata = validate)
yhat_2 = predict(lm.2, newdata = validate)
yhat_3 = predict(lm.3, newdata = validate)
yhat_4 = predict(lm.4, newdata = validate)
```

These vectors of fitted values can be used to compute the residuals for the validation observations, which in turn can be used to compute the *Cross-Validated Mean Square Error* (CV-MSE), a measure of out-of-sample performance. The CV-MSE is computed as:

$$\text{CV-MSE} = \frac{1}{n} \sum_{i=1}^{n} e_i^2$$

where n is the number of observations in the validation set. We can compute the CV-MSE for each of the candidate models.

```
sum((validate$life_expectancy - yhat_1) ^ 2) / nrow(validate)
```

```
## [1] 0.7349516
```

```
sum((validate$life_expectancy - yhat_2) ^ 2) / nrow(validate)
```

```
## [1] 0.7483797
```

```
sum((validate$life_expectancy - yhat_3) ^ 2) / nrow(validate)
```

```
## [1] 0.8547222
```

```
sum((validate$life_expectancy - yhat_4) ^ 2) / nrow(validate)
```

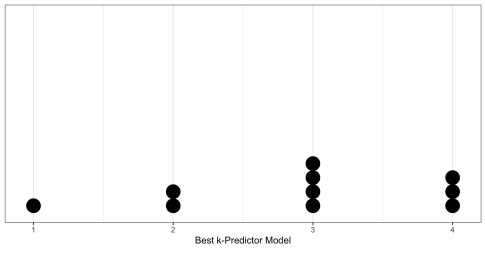
```
## [1] 0.8968336
```

The candidate model having the smallest CV-MSE is the model that should be adopted. In this case, the values for the CV-MSE suggest adopting the single predictor model.

Leave-One-Out Cross-Validation

There are two major problems with the cross-validation we just performed.

1. The estimate of the CV-MSE is highly dependent on the observations chosen to be in the training and validation sets. This is illustrated in the figure below which shows the CV-MSE values for the four models for 10 different random splits of the data. Some of these splits support adopting the quadratic polynomial model, other splits support adopting the cubic polynomial model, and one supports the linear model!



Model that has the lowest CV-MSE for 10 random splits of the data.

2. Only a subset of the observations (those in the training set) are used to initially fit the model. Since statistical methods tend to perform worse when trained on fewer observations, this suggests that the CV-MSE may tend to overestimate the error rate (model accuracy measure) for the model fit.

Leave-one-out cross-validation (LOOCV) is one method that can be used to overcome these issues. The algorithm for performing LOOCV is:

1. Hold out the ith observation as your validation data (a single observation) and use the remaining $n-1$ observations as your training data.
2. Fit all candidate models to the training data.
3. Use the estimated coefficients from those fits to compute MSE_i using the validation data.
4. Repeat Steps 2–4 for each observation.

We then have n estimates of the MSE that can be averaged to get the overall CV-MSE.

$$\text{CV-MSE} = \frac{1}{n}\sum_{i=1}^{n}\text{MSE}_i$$

Since the validation dataset is composed of a single observation ($n = 1$), MSE_i is simply the squared residual value for the validation observation.

$$\begin{aligned}\text{MSE} &= \frac{1}{n}\sum_{i=1}^{n}e_i^2 \\ &= \frac{1}{1}\sum_{i=1}^{n}e_i^2 \\ &= \sum_{i=1}^{n}e_i^2\end{aligned}$$

We can carry out LOOCV by using a `for()` loop to iterate through the algorithm for each of the n observations. Here is some example syntax:

```
# Set up empty vector to store results
mse_1 = rep(NA, 52)
mse_2 = rep(NA, 52)
mse_3 = rep(NA, 52)
mse_4 = rep(NA, 52)
mse_5 = rep(NA, 52)

# Loop through the cross-validation
for(i in 1:nrow(z_usa)){
  train = z_usa %>% filter(row_number() != i)
  validate = z_usa %>% filter(row_number() == i)

  lm.1 = lm(life_expectancy ~ -1 + income,
        data = train)
  lm.2 = lm(life_expectancy ~ -1 + income + population,
        data = train)
  lm.3 = lm(life_expectancy ~ -1 + income + population + illiteracy,
        data = train)
  lm.4 = lm(life_expectancy ~ -1 + income + population + illiteracy +
        murder, data = train)

  yhat_1 = predict(lm.1, newdata = validate)
  yhat_2 = predict(lm.2, newdata = validate)
  yhat_3 = predict(lm.3, newdata = validate)
  yhat_4 = predict(lm.4, newdata = validate)

  mse_1[i] = (validate$life_expectancy - yhat_1) ^ 2
  mse_2[i] = (validate$life_expectancy - yhat_2) ^ 2
  mse_3[i] = (validate$life_expectancy - yhat_3) ^ 2
  mse_4[i] = (validate$life_expectancy - yhat_4) ^ 2

}

# Compute CV-MSE
mean(mse_1)
```

```
## [1] 0.8319288
```

```
mean(mse_2)
```

```
## [1] 0.8068921
```

```
mean(mse_3)
```

```
## [1] 0.7686342
```

```
mean(mse_4)
```

```
## [1] 1.123998
```

The LOOCV results point toward the three-predictor model (smallest average CV-MSE), although the CV-MSE for the two-predictor model is not that much larger. Since this method is less biased; it trains the models on a much larger set of observations, its results are more believable than the simple cross-validation. Another advantage of LOOCV is that it will always produce the same results as opposed to simple cross-validation) since there is no randomness in producing the training and validation datasets.

LOOCV can be computationally expensive; we have to fit the candidate models n times. It turns out, however, that models fit with least squares linear or polynomial regression, can give us the LOOCV results using the following formula:

$$\text{CV-MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{e_i}{1-h_{ii}}\right)^2$$

where $\hat{y}_i$ is the ith fitted value from the original least squares fit, and h_{ii} is the leverage value. This is similar to the model (biased) MSE, except the ith residual is divided by $1-h_{ii}$.

LOOCV is a very general method, and can be used with any kind of modeling. For example we could use it with logistic regression, or mixed-effects analysis, or any of the methods you have encountered in your statistics courses to date. That being said, the formula does not hold for all these methods, so outside of linear and polynomial regression, the candidate models will actually need to be fitted n times.

k-Fold Cross-Validation

One alternative to LOOCV is k-fold cross-validation. The algorithm for k-fold cross-validation is:

1. Randomly divide the data into k groups or folds.
2. Hold out the ith fold as your validation data and use the remaining $k-1$ folds as your training data.
3. Fit all candidate models to the training data.

4. Use the estimated coefficients from those fits to compute MSE_i using the validation data.
5. Repeat Steps 2–4 for each fold.

We then have k estimates of the MSE that can be averaged to get the overall CV-MSE.

$$\text{CV-MSE} = \frac{1}{k}\sum_{i=1}^{k}\text{MSE}_i$$

From this algorithm, it is clear that LOOCV is a special case of k-fold cross-validation in which k is set to equal n. In practice we typically use $k = 5$ or $k = 10$.

We can carry out k-fold cross-validation by using the `crossv_kfold()` function from the **modelr** package. This function takes the argument `k=` to indicate the number of folds. Here is some example syntax to carry out a 10-fold cross-validation:

```
# Divide data into 10 folds
set.seed(100)

my_cv = z_usa %>%
  crossv_kfold(k = 10)
```

Then we will use the `map()` and `map2_dbl()` functions from the **purrr** package to fit a model to the training (`train`) data and find the MSE on the validation (`test`) data created from the `crossv_kfold()` function. We will have to carry this out for each of the candidate models.

> For more detailed information about using the purrr functions, see Jenny Bryan's fantastic purrr tutorial (https://jennybc.github.io/purrr-tutorial/index.html).

```
# Best 1-predictor model
cv_1 = my_cv %>%
  mutate(
    model = map(train, ~lm(life_expectancy ~ 1 + income, data = .)),
    MSE = map2_dbl(model, test, modelr::mse),
    k = 1
    )

# Best 2-predictor model
cv_2 = my_cv %>%
  mutate(
    model = map(train, ~lm(life_expectancy ~ 1 + income + population, data =
        .)),
    MSE = map2_dbl(model, test, modelr::mse),
    k = 2
    )

# Best 3-predictor model
cv_3 = my_cv %>%
  mutate(
    model = map(train, ~lm(life_expectancy ~ 1 + income + population +
        illiteracy, data = .)),
    MSE = map2_dbl(model, test, modelr::mse),
    k = 3
    )

# Best 4-predictor model
cv_4 = my_cv %>%
 mutate(
    model = map(train, ~lm(life_expectancy ~ 1 + income + population +
        illiteracy + murder, data = .)),
    MSE = map2_dbl(model, test, modelr::mse),
    k = 4
    )
```

Once we have the results, we can stack these into single data frame and then use `group_by()` and `summarize()` to obtain the CV-MSE estimates.

```
rbind(cv_1, cv_2, cv_3, cv_4) %>%
  group_by(k) %>%
  summarize(
    cv_mse = mean(MSE)
  )
```

```
## # A tibble: 4 x 2
##       k cv_mse
##   <dbl>  <dbl>
## 1     1  0.885
## 2     2  0.859
## 3     3  0.830
## 4     4  1.22
```

The results of carrying out the 10-fold cross-validation suggest that we adopt the best two- or three-predictor model.

Using k-fold cross-validation is computationally less expensive so long as $k < n$. But this computational gain has a cost in that the results are again dependent on the k random splits. This variation, however, is less than that in the single split simple cross-validation. We can also alleviate some of this by fitting the k-fold cross-validation several times and averaging across the results to get the CV-MSE estimate.

There are several R packages that will fit a k-fold cross-validation (e.g., DAAG).

Model Selection

The different methods of selecting variables suggest we adopt the following models:

Predictors selected from forward selection (FS), backward elimination (BE) and cross-validation (CV) methods. The CV-MSE was used as a performance metric for model selection in all cross-validation methods.

Method	Model
FS (t-value)	Income, Population, Illiteracy, Murder
FS (AIC)	Income, Population, Illiteracy
BE (R2)	Income, Population, Illiteracy
All Subsets (AICc)	Income, Population, Illiteracy

Method	Model
Simple CV	Income
Leave-One-Out CV	Income, Population, Illiteracy
10-Fold CV	Income, Population, Illiteracy

> While the different CV methods can suggest different models, this is usually less problematic when the sample size is larger; we only have 52 observations in the `z_usa` data.

The LOOCV and k-fold CV are better suited for considering how the model will perform on future data sets, so I would adopt the model that includes income, population, and illiteracy rate. To get the coefficient estimates, we fit whichever model we adopt to the FULL data set (with all the observations) as this will give us the "best" estimates.

```
# Fit model
lm.3 = lm(life_expectancy ~ -1 + income + population + illiteracy, data =
        z_usa)

# Get model-level output
glance(lm.3)
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic p.value    df logLik   AIC
BIC
##       <dbl>         <dbl> <dbl>     <dbl>   <dbl> <dbl>  <dbl> <dbl>
<dbl>
## 1     0.349         0.310 0.823      8.77 9.24e-5     3  -62.1  132.
140.
## # … with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

```
# Get coefficient-level output
tidy(lm.3)
```

```
## # A tibble: 3 x 5
##   term       estimate std.error statistic  p.value
##   <chr>         <dbl>     <dbl>     <dbl>    <dbl>
## 1 income        0.488     0.115      4.23 0.000102
## 2 population    0.392     0.145      2.70 0.00944
## 3 illiteracy   -0.310     0.145     -2.13 0.0379
```

Reporting Results from a Cross-Validation

When we report the results of a regression model evaluated using cross-validation, there are some subtle differences in what is reported. At the model-level, we report the R^2 from the adopted model fitted to the full-data. We also report the CV-MSE as a measure of the model error for future observations (generalized error).

At the coefficient-level we typically report the coefficient estimates and standard errors based on fitting the adopted model to the full data. However, we DO NOT REPORT NOR INTERPRET P-VALUES. The p-values do not take into account that the model was selected using cross-validation. Because of this they are incredibly misleading. As with any model, interpretations should also be offered, again typically by way of providing a plot of the fitted model to help facilitate these interpretations. For example:

The regression model using income, population, and illiteracy rate to predict variation in life expectancy was selected using 10-fold cross-validation (CV-MSE = 0.830). The model explains 34.9% of the variation in life expectancies. The fitted equation is:

$$\hat{\text{Life Expectancy}}_i = 0.488(\text{Income}_i) + 0.392(\text{Population}_i) - 0.310(\text{Illiteracy Raten}_i)$$

where, the outcome and all predictors are standardized. This model suggests moderate, positive, partial effects of both income and population and a moderate negative effect of illiteracy rate on life expectancy.

Information Criteria and Cross-Validation

In EPsy 8252, we learned about using information criteria for model selection. In particular, we used AIC, BIC, and AICc for model selection. It turns out the AIC-based model selection and cross-validation are asymptotically equivalent Stone (1977). For linear models, using BIC for model selection is symptotically equivalent to leave-v-out cross-validation when $v = n\left(1 - \frac{1}{\ln(n) - 1}\right)$ (Shao, 1997).

As a practical note, computer simulations have suggested that the results from using AICc for model selection will, on average, be quite similar to those from cross-validation techniques. As such, AICc can be a useful alternative to the computationally expensive cross-validation methods. However, using AICc does not provide a measure of the model performance (MSE) in new datasets like cross-validation does.

```
# Fit models to all data
lm.1 = lm(life_expectancy ~ -1 + income,
        data = z_usa)
lm.2 = lm(life_expectancy ~ -1 + income + population,
        data = z_usa)
lm.3 = lm(life_expectancy ~ -1 + income + population + illiteracy,
        data = z_usa)
lm.4 = lm(life_expectancy ~ -1 + income + population + illiteracy + murder,
        data = z_usa)


# Load library
library(AICcmodavg)

# Get AICc for all models
aictab(
  cand.set = list(lm.1, lm.2, lm.3, lm.4),
  modnames = c("Best 1-Predictor", "Best 2-Predictor", "Best 3-Predictor",
        "Best 4-Predictor")
)
```

```
##
## Model selection based on AICc:
##
##                   K   AICc Delta_AICc AICcWt Cum.Wt     LL
## Best 3-Predictor 4 133.05       0.00   0.48   0.48 -62.10
## Best 4-Predictor 5 134.25       1.20   0.26   0.74 -61.47
## Best 2-Predictor 3 135.32       2.27   0.15   0.89 -64.41
## Best 1-Predictor 2 136.02       2.97   0.11   1.00 -65.89
```

The model with the lowest AICc is, indeed, the three-predictor model. There is also some evidence for the four-predictor model given the AICc weight.

References

Shao, J. (1997). An asymptotic theory for linear model selection. *Statistica Sinica*, *7*, 221–264.

Stone, M. (1977). An asymptotic equivalence of choice of model by cross-validation and akaike's criterion. *Journal of the Royal Statistical Society, Series B*, *39*, 44–47.