Unit 2 Lecture 3: Cross-validation

September 23, 2021

In this R demo, we will implement cross-validation to select the degrees of freedom of a natural spline fit, using the running example from the previous class.

Training and validation

Let us create a training set:

```
set.seed(1)
f = function(x)(sin(3*x))
n = 50
sigma = 1
train_data = tibble(x = seq(0, 2*pi, length.out = n),
                    y = f(x) + rnorm(n, sd = sigma))
train_data
```

```
## # A tibble: 50 x 2
##        x      y
##    <dbl>  <dbl>
##  1 0     -0.626
##  2 0.128  0.559
##  3 0.256 -0.140
##  4 0.385  2.51
##  5 0.513  1.33
##  6 0.641  0.118
##  7 0.769  1.23
##  8 0.898  1.17
##  9 1.03   0.640
## 10 1.15  -0.620
## # ... with 40 more rows
```

Let's also suppose we have a large validation set on our hands:

```
N = 50000
validation_data = tibble(x = seq(0, 2*pi, length.out = N),
                    y = f(x) + rnorm(n, sd = sigma))
```

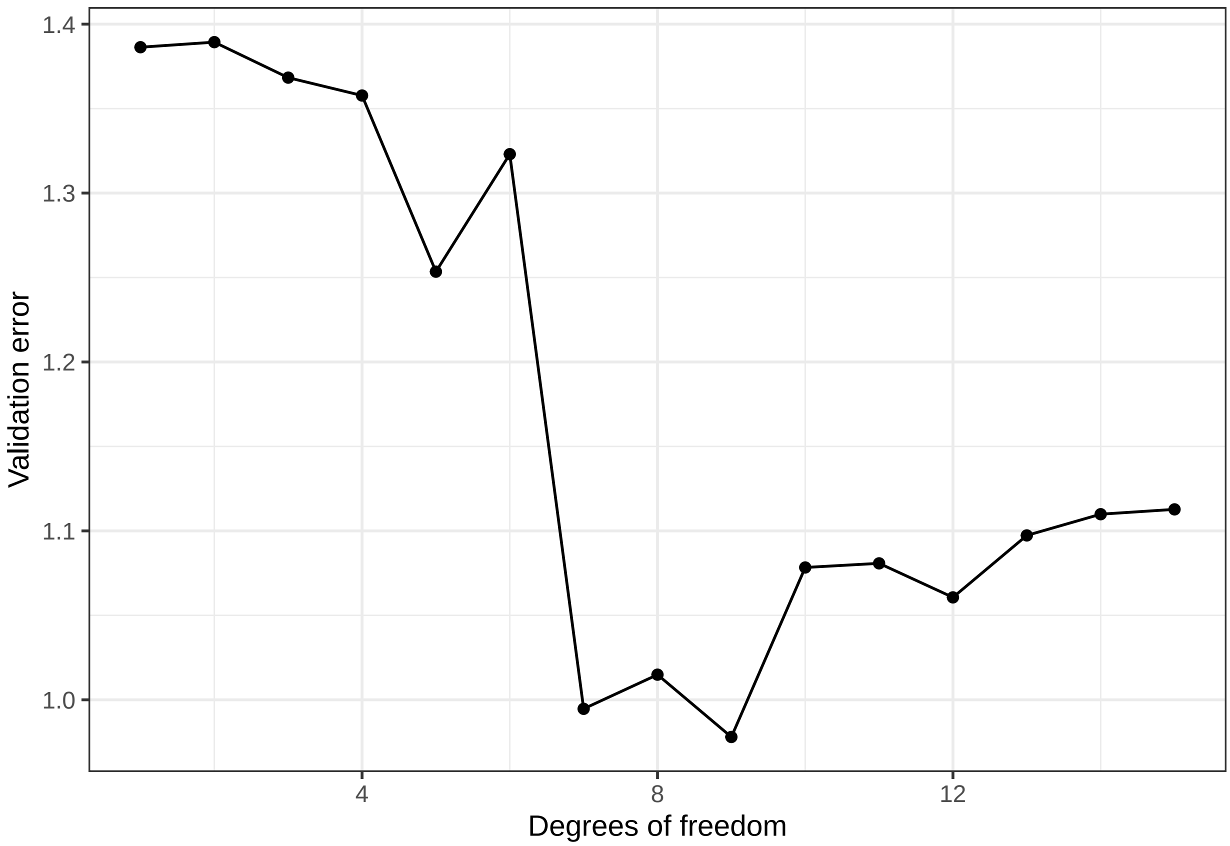
Now let's fit splines with `df` = 1,2,. . . ,15 to the training data, and evaluate their test error using the test set:

```
# compute the validation error
max_df = 15
validation_error = numeric(max_df)
df = 1
for(df in 1:max_df){
  formula = sprintf("y ~ splines::ns(x, df = %d)", df)
  spline_fit = lm(formula = formula, data = train_data)
  y_hat_validation = predict(spline_fit, newdata = validation_data)
  validation_error[df] = validation_data %>%
```

```
    cbind(y_hat_validation) %>%
    summarise(mean((y_hat_validation-y)^2)) %>%
    pull()
}
validation_error
```

```
##  [1] 1.3863485 1.3893519 1.3683364 1.3577412 1.2534633 1.3230281 0.9946587
##  [8] 1.0148754 0.9779877 1.0783341 1.0807794 1.0606353 1.0973088 1.1098777
## [15] 1.1127368
```

```
# plot the validation error
p_val = tibble(df = 1:max_df, validation_error) %>%
  ggplot(aes(x = df, y = validation_error)) +
  geom_point() + geom_line() +
  xlab("Degrees of freedom") +
  ylab("Validation error") + theme_bw()
plot(p_val)
```



The issue is that we usually do not have a giant validation set for model selection purposes. We need to make do with our smallish training set for both model training and model selection. This is where cross-validation comes in handy!

Cross-validation for df = 5

The idea is to split our training samples into *folds* and then have the folds take turns being the validation set. Let's take a look.

```r
K = 10
folds = sample(rep(1:K, n/K))
train_data = train_data %>% bind_cols(tibble(fold = folds))
train_data
```

```
## # A tibble: 50 x 3
##        x      y  fold
##    <dbl>  <dbl> <int>
##  1 0     -0.626     3
##  2 0.128  0.559     6
##  3 0.256 -0.140     4
##  4 0.385  2.51      2
##  5 0.513  1.33      9
##  6 0.641  0.118     7
##  7 0.769  1.23      8
##  8 0.898  1.17      1
##  9 1.03   0.640     1
## 10 1.15  -0.620     5
## # ... with 40 more rows
```

Question: How would we select the data in fold number 1? How would we select all the data except fold number 1?

Let's first use cross-validation to estimate the test error for a spline fit with 5 degrees of freedom.

```r
# create a vector of out-of-fold predictions
out_of_fold_predictions = numeric(n)

# iterate over folds
for(current_fold in 1:K){
  # out-of-fold data will be used for training
  out_of_fold_data = train_data %>% filter(fold != current_fold)
  # in-fold data will be used for validation
  in_fold_data = train_data %>% filter(fold == current_fold)

  out_of_fold_data
  in_fold_data

  # train on out-of-fold data
  spline_fit = lm(y ~ splines::ns(x, df = 5), data = out_of_fold_data)

  # predict on in-fold data
  out_of_fold_predictions[folds == current_fold] =
    predict(spline_fit, newdata = in_fold_data)
}

# add the out-of-fold predictions to the data frame
results = train_data %>%
  bind_cols(yhat = out_of_fold_predictions)
results
```

```
## # A tibble: 50 x 4
##        x      y  fold   yhat
##    <dbl>  <dbl> <int>  <dbl>
##  1 0     -0.626     3  1.85
##  2 0.128  0.559     6  0.726
```

```
##   3 0.256 -0.140     4  0.878
##   4 0.385  2.51      2  0.252
##   5 0.513  1.33      9  0.242
##   6 0.641  0.118     7  0.181
##   7 0.769  1.23      8 -0.0212
##   8 0.898  1.17      1 -0.374
##   9 1.03   0.640     1 -0.490
##  10 1.15  -0.620     5  0.0871
## # ... with 40 more rows
```

```
# compute the CV estimate and standard error
results %>%
  group_by(fold) %>%
  summarise(cv_fold = mean((yhat-y)^2)) %>%  # CV estimates per fold
  summarise(cv_mean = mean(cv_fold),
            cv_se = sd(cv_fold)/sqrt(K))
```

```
## # A tibble: 1 x 2
##   cv_mean cv_se
##     <dbl> <dbl>
## 1    1.70 0.339
```

What are two reasons this CV estimate may be different from the validation error estimated above?

Cross-validation for df = 1,2,. . . ,15

Now let's repeat what we did above for many degrees of freedom, because after all, the point of cross-validation is to choose the degrees of freedom.

```
# create a matrix for out-of-fold predictions
out_of_fold_predictions = matrix(0, n, max_df) %>%
  as_tibble() %>%
  setNames(paste0('y_hat_', 1:max_df))
```

```
## Warning: The `x` argument of `as_tibble.matrix()` must have unique column names if `.name_repair` is
## Using compatibility `.name_repair`.
```

```
# iterate over folds
for(current_fold in 1:K){
  # out-of-fold data will be used for training
  out_of_fold_data = train_data %>% filter(fold != current_fold)
  # in-fold data will be used for validation
  in_fold_data = train_data %>% filter(fold == current_fold)

  # iterate over df
  for(df in 1:15){
    # train on out-of-fold data
    formula = sprintf("y ~ splines::ns(x, df = %d)", df)
    spline_fit = lm(formula = formula, data = out_of_fold_data)

    # predict on in-fold data
    out_of_fold_predictions[folds == current_fold, df] =
      predict(spline_fit, newdata = in_fold_data)
  }
}
```

```r
# add the out-of-fold predictions to the data frame
results = train_data %>% bind_cols(out_of_fold_predictions)
results
```

```
## # A tibble: 50 x 18
##        x      y  fold y_hat_1 y_hat_2 y_hat_3 y_hat_4 y_hat_5 y_hat_6 y_hat_7
##    <dbl>  <dbl> <int>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
##  1 0     -0.626     3   0.601   0.853  0.928    0.989  1.85     1.74     0.692
##  2 0.128  0.559     6   0.659   0.615  0.559    0.424  0.726    0.758    0.394
##  3 0.256 -0.140     4   0.435   0.590  0.608    0.714  0.878    0.789    0.712
##  4 0.385  2.51      2   0.252   0.190  0.225    0.253  0.252    0.245    0.603
##  5 0.513  1.33      9   0.298   0.349  0.355    0.375  0.242    0.284    0.947
##  6 0.641  0.118     7   0.421   0.408  0.438    0.432  0.181    0.345    1.38
##  7 0.769  1.23      8   0.318   0.341  0.340    0.349 -0.0212   0.0112   1.05
##  8 0.898  1.17      1   0.229   0.255  0.236    0.241 -0.374   -0.665    0.319
##  9 1.03   0.640     1   0.219   0.238  0.204    0.206 -0.490   -0.761    0.145
## 10 1.15  -0.620     5   0.222   0.244  0.0744   0.302  0.0871   0.335    0.392
## # ... with 40 more rows, and 8 more variables: y_hat_8 <dbl>, y_hat_9 <dbl>,
## #   y_hat_10 <dbl>, y_hat_11 <dbl>, y_hat_12 <dbl>, y_hat_13 <dbl>,
## #   y_hat_14 <dbl>, y_hat_15 <dbl>
```

```r
# compute the CV estimate and standard error
cv_error = results %>%
  pivot_longer(-c(x,y,fold),
               names_to = "df",
               names_prefix = "y_hat_",
               names_transform = list(df = as.integer),
               values_to = "yhat") %>%
  group_by(df, fold) %>%
  summarise(cv_fold = mean((yhat-y)^2)) %>%  # CV estimates per fold
  summarise(cv_mean = mean(cv_fold),
            cv_se = sd(cv_fold)/sqrt(K))
```

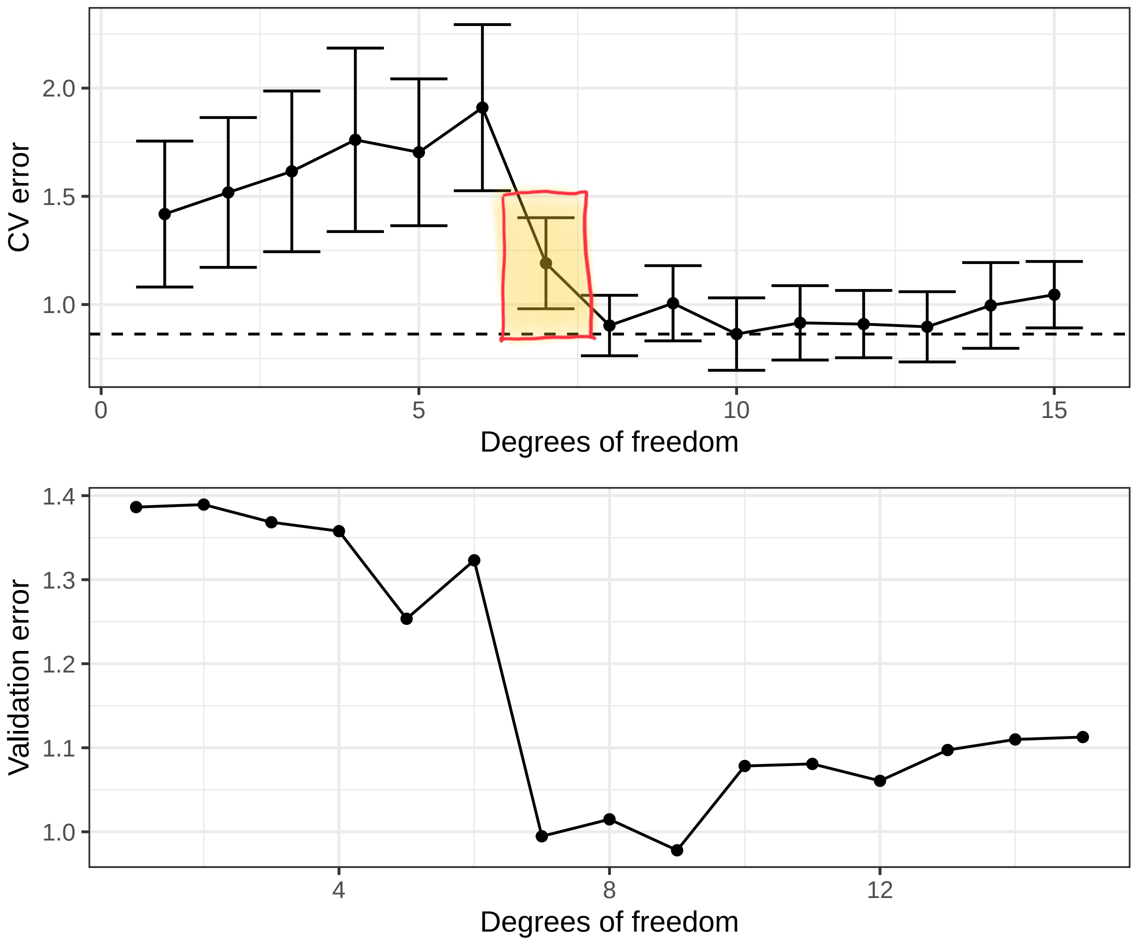
```
## `summarise()` has grouped output by 'df'. You can override using the `.groups` argument.
```

```r
cv_error
```

```
## # A tibble: 15 x 3
##       df cv_mean cv_se
##    <int>   <dbl> <dbl>
##  1     1   1.42  0.337
##  2     2   1.52  0.346
##  3     3   1.62  0.371
##  4     4   1.76  0.424
##  5     5   1.70  0.339
##  6     6   1.91  0.384
##  7     7   1.19  0.210
##  8     8   0.903 0.140
##  9     9   1.01  0.174
## 10    10   0.863 0.167
## 11    11   0.915 0.172
## 12    12   0.910 0.155
## 13    13   0.897 0.162
## 14    14   0.996 0.198
## 15    15   1.05  0.153
```

```
# plot the results, along with the previously computed validation error
p_cv = cv_error %>%
  ggplot(aes(x = df, y = cv_mean, ymin = cv_mean-cv_se, ymax = cv_mean+cv_se)) +
  geom_point() + geom_line() + geom_errorbar() +
  geom_hline(aes(yintercept = min(cv_mean)), linetype = "dashed") +
  xlab("Degrees of freedom") + ylab("CV error") +
  theme_bw()

cowplot::plot_grid(p_cv, p_val, nrow = 2)
```



Based on the one-standard-error rule, what degrees of freedom would we select based on the cross-validation?

Let's wrap this cross-validation procedure into a function:

```
cross_validate_spline = function(x, y, nfolds, df_values){
  # a few checks of the inputs
  stopifnot(is.vector(x))
  stopifnot(is.vector(y))
  stopifnot(length(x) == length(y))

  # divide training data into folds
  n = length(x)
  train_data = tibble(x,y)
  folds = sample(rep(1:nfolds, length.out = n))
  train_data = train_data %>% mutate(fold = folds)
```

takes 1 out
to test on and
all other folds
used for train

```r
# create a matrix for out-of-fold predictions
num_df_values = length(df_values)
out_of_fold_predictions =
  matrix(0, n, num_df_values) %>%
  as_tibble() %>%
  setNames(paste0('y_hat_', df_values))

# iterate over folds
for(current_fold in 1:nfolds){
  # out-of-fold data will be used for training
  out_of_fold_data = train_data %>% filter(fold != current_fold)
  # in-fold data will be used for validation
  in_fold_data = train_data %>% filter(fold == current_fold)

  # iterate over df
  for(i in 1:num_df_values){
    df = df_values[i]

    # train on out-of-fold data
    formula = sprintf("y ~ splines::ns(x, df = %d)", df)
    spline_fit = lm(formula = formula, data = out_of_fold_data)

    # predict on in-fold data
    out_of_fold_predictions[folds == current_fold, i] =
      predict(spline_fit, newdata = in_fold_data)
  }
}

# add the out-of-fold predictions to the data frame
results = train_data %>% bind_cols(out_of_fold_predictions)
results

# compute the CV estimate and standard error
cv_table = results %>%
  pivot_longer(-c(x,y,fold),
               names_to = "df",
               names_prefix = "y_hat_",
               names_transform = list(df = as.integer),
               values_to = "yhat") %>%
  group_by(df, fold) %>%
  summarise(cv_fold = mean((yhat-y)^2)) %>%  # CV estimates per fold
  summarise(cv_mean = mean(cv_fold),
            cv_se = sd(cv_fold)/sqrt(nfolds))

df.1se = cv_table %>%
  filter(cv_mean-cv_se <= min(cv_mean)) %>%
  summarise(min(df)) %>%
  pull()

df.min = cv_table %>%
  filter(cv_mean == min(cv_mean)) %>%
  summarise(min(df)) %>%
  pull()
```

```r
  # plot the results, along with the previously computed validation error
  cv_plot = cv_table %>%
    ggplot(aes(x = df, y = cv_mean, ymin = cv_mean-cv_se, ymax = cv_mean+cv_se)) +
    geom_point() + geom_line() + geom_errorbar() +
    geom_hline(aes(yintercept = min(cv_mean)), linetype = "dashed") +
    xlab("Degrees of freedom") + ylab("CV error") +
    theme_bw()

  # return CV table and plot
  return(list(cv_table = cv_table,
              cv_plot = cv_plot,
              df.1se = df.1se,
              df.min = df.min))
}

out = cross_validate_spline(train_data$x,
                            train_data$y,
                            nfolds = 10,
                            df_values = 1:15)
```

```
## `summarise()` has grouped output by 'df'. You can override using the `.groups` argument.
```

A copy of this function is stored at `stat-471-fall-2021/functions/cross_validate_spline.R`. You'll use it in Homework 2. To "load" the function into your workspace, you need to `source` the above file.

Exercise: Use cross-validation and the one-standard error rule to select the optimal number of degrees of freedom for regressing `wage` on `age` in the `Wage` data from the `ISLR2` package. Make the CV plot, and produce a scatter plot of `wage` versus `age` with the optimal spline fit superimposed.

```r
wage_data = ISLR2::Wage %>% as_tibble()
```