Model Selection & Validation

Julian D. Karch

Model Validation

- Setup: Predict outcome Y using predictors X
- **Model Validation**: How well does a particular model perform in the population when training on a particular training set (size, population)?
- Example: How well does polynomial regression with degree 15 predict continuous outcome Y (trained on training set of size 1000)

Optimum

- Data cheap $\rightarrow$ can get large training and test set from population
- Train model on training set, results in $\hat{f}$
- Estimate performance of $\hat{f}$ on test set set

Example Population

These functions represent our population

```
true_f <- function(x) {
  8 * sin(x)
}

gen_data <- function(n, true_f) {
  x <- runif(n, min = -3, max = 3)
  y <- true_f(x) + rnorm(n, sd = 1)
  data.frame(x = x, y = y)
}
```

Therefore, in our population

$$Y = 8\sin(X) + \epsilon, \text{with } X \sim \mathcal{U}(-3, 3) \text{ and } \epsilon \sim \mathcal{N}(0, 1)$$

Question: Given this population, what is the irreducible error? Thus, what is the lowest MSE any model can achieve on the population?

Optimum Demonstration

- Gather training set from population of certain size $(1,000)$
- Train model on training set
- Gather test set from population of large size $(1,000,000)$
- Estimate performance on test set

Optimum in R

```r
# gather training set
set.seed(123)
train_set <- gen_data(10^3, true_f)
# train model
fitted_model <- lm(y ~ poly(x, 15), data = train_set)
# gather test set
test_set <- gen_data(10^6, true_f)
# estimate performance
predictions <- predict(fitted_model, test_set)
true_mse <- mean((predictions - test_set$y)^2)
print(true_mse)
```

```
## [1] 1.013873
```

We can treat this (1.01) as true value.

Problem

- Data expensive/hard to get. We often do not have a (large) test set
- Typical: have one training set (of size $1,000$); want to know performance of a method trained on this training set.
- Naive solution: split available data set into training and test set

Train-Test Split Procedure

- Gather data set from population of certain size ($1,000$)
- Split into training and test (for example, 50% each)
- Train model on training set
- Estimate performance on test set

Train-Test Split Code

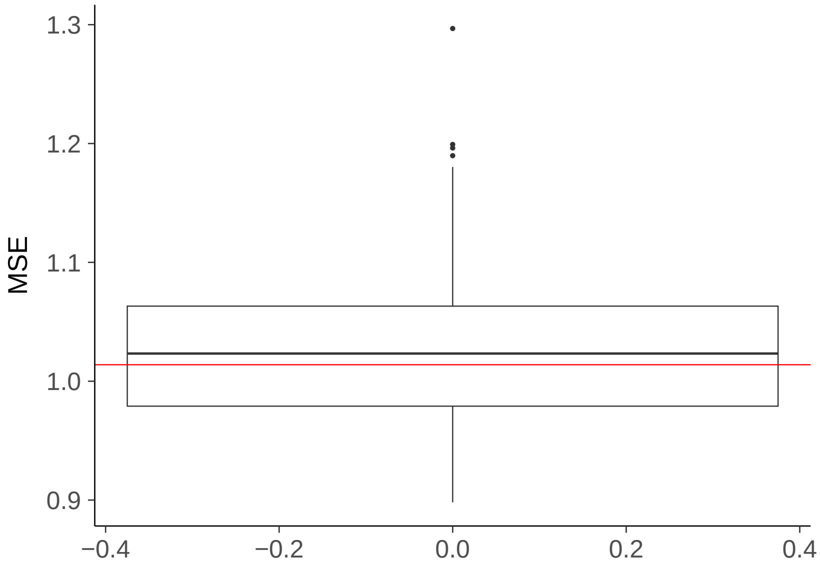
```r
train_test <- function(prob_train) {
  data_set <- gen_data(10^3, true_f) # obtain data set
  # split into training and test
  train_size <- floor(prob_train * nrow(data_set))
  train_ind <- sample(seq_len(nrow(data_set)),
    size = train_size
  )
  train_set <- data_set[train_ind, ]
  test_set <- data_set[-train_ind, ]
  # train on training set
  fitted_model <- lm(y ~ poly(x, 15), data = train_set)
  # estimate MSE on test set
  predictions <- predict(fitted_model, test_set)
  test_mse <- mean((predictions - test_set$y)^2)
  return(test_mse)
}
```

Train-Test Split Conclussion

```r
res <- train_test(0.5)
```

- Obtained MSE `res`=1.06 is substantially larger than true MSE of 1.01. Let's see whether this is a consistent pattern.

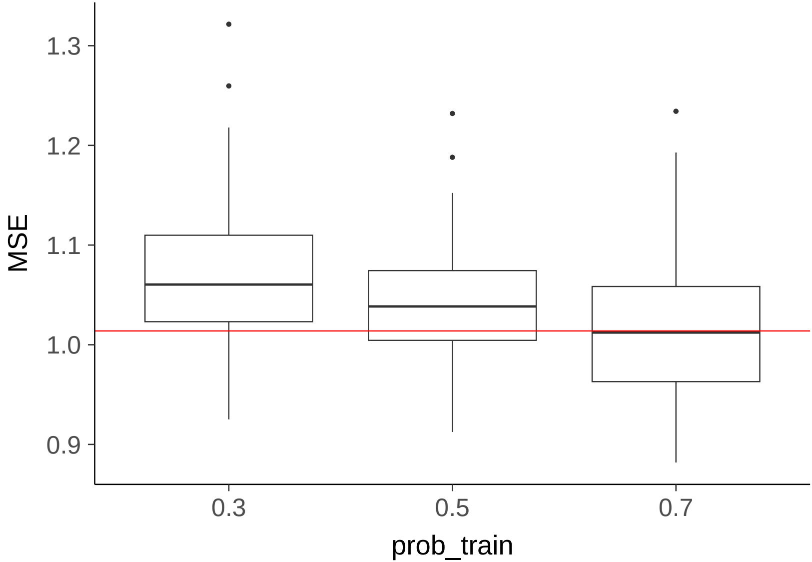
Bias and Variance of Train Test Split



Bias and Variance of Train Test Split

- Training set size smaller (500) compared to full training set (1000) $\rightarrow$ error is overestimated $\rightarrow$ also rather large variance in our estimation of $(\hat{f})$
- Test set size is rather small $\rightarrow$ rather large variance in our estimation of MSE
- Let's have a look at different allocations of test and training set

Different Train-Test Splits



Different Train-Test Splits

- With larger training set size
  - Bias goes down
  - Variance of $\hat{f}$ goes down
  - Variance of MSE estimates goes up ✓ test set small
  - Total variance can go up or down but tends to go up when test set is very small
- $\rightarrow$ no good option available
- Fundamental problem: data point is used either for training OR testing

Question

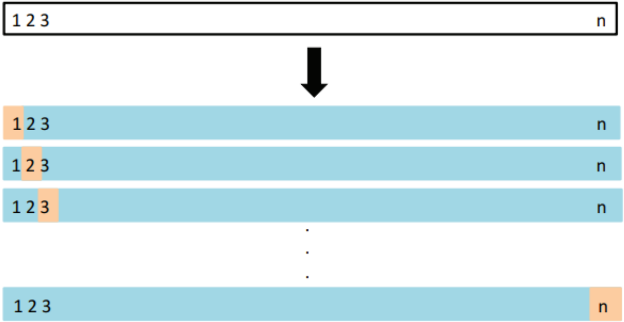
You have a data set of size $1,000$ available and want to know how well linear regression predicts in the population when trained with this full data set. You estimate this by splitting the full data set into two halves, a training and a test set. Do you likely over- or underestimate the MSE of your model?

overestimate mse

Cross-Validation

Leave-one-out Cross-validation

- Cross-validation: Use each data point for training AND testing
- Leave-one-out cross-validation procedure:
  - Leave out the first observation
  - Train on all other observations
  - Predict for the first observation
  - Save this as the test set prediction for the first observation
  - Repeat for all observations
  - Compute test set MSE as always



Leave-one-out Cross-validation Code

```
data_set <- train_set
predictions <- numeric(nrow(data_set))
for (i in 1:nrow(data_set)) {
  train_set <- data_set[-i, ]
  test_set <- data_set[i, ]
  fitted_model <- glm(y ~ poly(x, 15), data = train_set)
  predictions[i] <- predict(fitted_model, test_set)
}
loocv_mse <- mean((predictions - data_set$y)^2)
print(loocv_mse)
```

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

Estimated value `loocv_mse`=1.02 is pretty close to true value of 1.01

Leave-one-out Cross-validation Disadvantages

- Have to train our method n times
- $\rightarrow$ can be computationally costly
- More to come

k-fold Cross-validation optimize hyperparameters

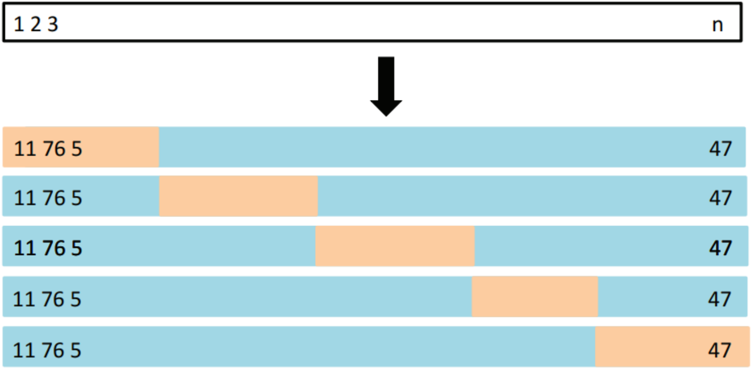
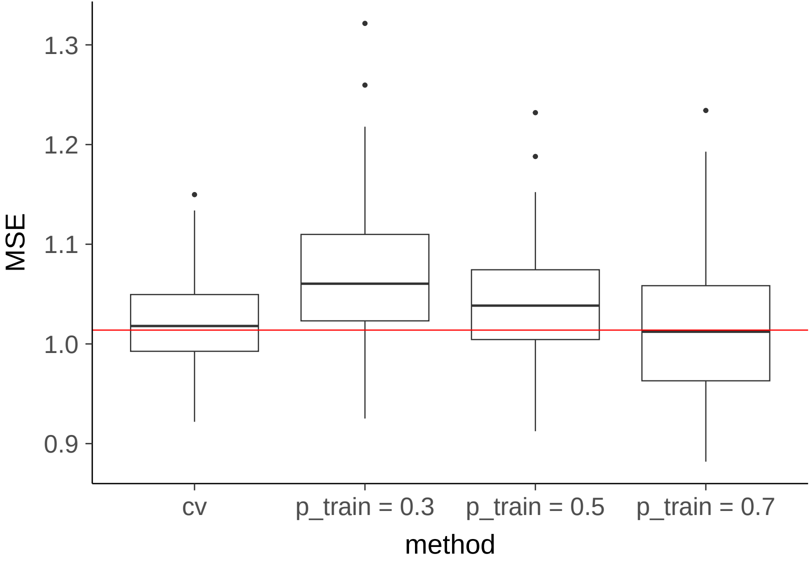


Figure 2: 5-fold cross-validation

Note that leave-one-out cross-validation is a special case with $k = n$.

10-fold Cross-validation vs Train-test Split



Cross-validation with $k = 10$ is more accurate than all train-test split approaches (in this case but also typically)

Choosing k

- Bias: Depends on the size of the training set (larger is better).
  - Training on less data results in overestimation of test error.
  - Leave-one-out cross-validation has a small bias. $k=n$
  - Smaller k (smaller training sets) result in larger bias.
- Variance:
  - Smaller k results in smaller variance (see book for reason).
  - There is once again a bias-variance tradeoff.
  - In practice, 10- and 5-fold cross-validation tend to give a good compromise.

Question

Ignoring computational issues, why do we not always use leave-one-out cross-validation?

higher variance   test set is small

Model Selection

Model Selection + Validation

- So far: Model Validation: Estimate the performance of ONE model
- Practice:
  - Try out a few different models (e.g., polynomials of different degrees)
  - Model Selection: Select the one that predicts best (e.g., polynomial of a certain degree)
  - Estimate performance of selected model (e.g., MSE of selected polynomial)

Naive Approach

Naive idea: Use cross-validation for selection + validation:

- Obtain training set, decide on a set of candidates models (say polynomials of degree $1, 2, 3, \ldots, 25$)
- Estimate the performance of all candidate models using cross-validation
- Select best performing model according to cross-validation
- Use MSE estimated by cross-validation as performance estimate

Naive Approach in R

```r
#candidate models
degrees <- seq(from = 1, to = 25, by = 1)
cv_mses <- numeric(length(degrees))
# obtain training set
train_set <- gen_data(200, true_f)
#estimate performance of all candidate models with CV
for (i in 1:length(degrees)) {
  fitted_model <- glm(y ~ poly(x, degrees[i]),
                      data = train_set)
  cv_mses[i] <- cv.glm(train_set, fitted_model,
                       K = 10)$delta[1]
}
```

Estimated MSE of best model is `min(cv_mses)`=0.95.

Comparison True Value

```r
selected_degree <- which.min(cv_mses)
test_set <- gen_data(10^6, true_f)
fitted_model <- glm(y ~ poly(x, selected_degree),
                    data = train_set)
predictions <- predict(fitted_model, test_set)
true_mse_sel <- mean((predictions - test_set$y)^2)
```

- True MSE of best model is `true_mse_sel`=1.1, which is considerably higher than estimated MSE of 0.95.
- What is happening?

A Simpler Equivalent Example

- Want to estimate how often a die shows six (we know it is $\frac{1}{6} \approx 0.17$).
- Throw a die 100 times. Count how many sixes
- Repeat often and average:

```
reps <- 10^4
prob_6 <- numeric(reps)
for (i in 1:reps) {
  dice_throws <- sample(1:6, size = 100, replace = TRUE)
  prob_6[i] <- mean(dice_throws == 6)
}
print(mean(prob_6))
```

```
## [1] 0.17
```

- This works

Many Dice Procedure

- Repeat the same but there are potentially different dice and we want to know the dice with highest probability of 6s
- So, we throw 100 dice 100 times each. Count how many sixes. Select highest amount of sixes.
- Repeat often and average
- Twist: all dice are actually the same. So true value is still 0.17

Many dice R

```r
reps <- 10^3
prob_6 <- numeric(reps)
for (i in 1:reps) {
  best_dice <- 0
  for (dice in 1:100) {
    dice_throws <- sample(1:6, size = 100, replace = TRUE)
    prob <- mean(dice_throws == 6)
    if (prob > best_dice)
      best_dice <- prob
  }
  prob_6[i] <- best_dice
}
print(mean(prob_6))
```

```
## [1] 0.25
```

- This thus does not work

Many Dice Explanation

- Example: if we draw enough dice, one will be lucky enough to show much more 6s than it should
- Prediction: All our estimates have variance, by selecting among many estimates we are likely to select one that was "lucky" (error estimate is much lower than its true error)
- In other words, we are likely to select a model that was particularly good on the available data but is not as good on the remaining data
- Thus, the model selection process itself also overfits!

Solution

- Example: Take the winning die and throw it again
- Prediction: Use three data sets:
  - Training Set (Train Models)
  - Validation Set (Select a Model)
  - Test Set (Estimate Performance of selected Model ONCE AT THE END)

Figure 3: Train-Validate-Test

Train-Validate-Test Procedure

- For each model
  - Train on training set
  - Estimate performance on validation set
- Select model with best performance on validation set
- Retrain model on training + validation set
- Estimate performance of retrained model on test set

Train-Validate-Test in R

```
set.seed(123)
## generate data sets
train_set <- gen_data(10^3, true_f)
validate_set <- gen_data(10^4, true_f)
test_set <- gen_data(10^4, true_f)

## estimate performance for each degree on validation set
degrees <- c(2, 5, 10)
mse_validate <- numeric(length(degrees))
for (i in 1:length(degrees)) {
  fitted_model <- lm(y ~ poly(x, degrees[i]),
                     data = train_set)
  predictions_validate <- predict(fitted_model,
                                  validate_set)
  mse_validate[i] <- mean((predictions_validate -
                             validate_set$y)^2)
}
```

Train-Validate-Test in R

```r
## select best model
best_model <- degrees[which.min(mse_validate)]
#retrain
train_validate <- rbind(train_set,
                        validate_set)
fitted_model_full <- lm(y ~ poly(x, best_model),
                        data = train_validate)
#estimate performance of best model on test set
predictions <- predict(fitted_model_full, test_set)
test_mse <- mean((predictions - test_set$y)^2)
```

Polynomial of degree `best_model`=5 was selected; estimated MSE of this model is `test_mse`=0.99

Train-Validate-Test Split

- Again in reality we typically only have one data set
- Can split available data into train, validate, test
- However, inefficient use of data, each observation is only used for one of the three

Cross-Validation + Test Set

- A commonly used approach is to use cross-validation for model selection and a test set for estimating the performance of the selected model
- Procedure:
  - Obtain data set, and split into training and test set
  - Put test set into a "safe"
  - Select a set of candidate models
  - Use cross-validation on the training set for model selection
  - Retrain selected model on training set
  - Get test set from the safe and estimate performance of selected final model (ONLY ONCE)

Cross-Validation + Test Set in R

```
data_set <- gen_data(10^3, true_f)
train_set <- data_set[1:800,]
test_set <- data_set[801:10^3,]

## estimate performance for each degree using
## cv on training set
degrees <- c(2, 5, 10)
mse_validate <- numeric(length(degrees))
for (i in 1:length(degrees)) {
  fitted_model <- glm(y ~ poly(x, degrees[i]),
                      data = train_set)
  mse_validate[i] <- cv.glm(train_set,
                            fitted_model)$delta[1]
}
```

Cross-validation + Test in R

```
## select best model, retrain
best_model <- degrees[which.min(mse_validate)]
fitted_model_full <- glm(y ~ poly(x, best_model),
                         data = train_set)
## estimate performance best model on test set
predictions <- predict(fitted_model_full, test_set)
test_mse <- mean((predictions - test_set$y)^2)
```

Polynomial of degree `best_model`=10 was selected; estimated MSE of this model is `test_mse`=1

Question

To optimally use the data, each observation should be used for training, validation, and testing. Is the cross-validation + test set approach using the data optimally?

Outlook and Book

Outlook

- It is possible to also use cross-validation instead of the test set
  - leads to nested cross-validation (optional reading: https://cran.r-project.org/web/packages/nestedcv/vignettes/nestedcv.html)
- There are more ways to select a model (optional reading 7.3-7.9 in ESL)
- Many variants of cross-validation exist, popular examples are stratified and repeated cross-validation

keep class distribution same as original data set.

What to Focus on in the Books

- Why smaller k leads to less variance (in ISLR)
- The right and wrong way to do cross-validation (was in preparatory videos and is in section 7.10 of ESL)
- What cross-validation really estimates: I pretended it is performance of fixed $\hat{f}$, this is fine but not entirely true (see ESL; this topic will be covered in detail in the advanced statistical learning course next semester)