Machine Learning in Public Health

Lecture 5: CV and Boostrap (Lab)

Dr. Yang Feng

Cross-Validation

The validation set approach

```
library(MASS)
library(tidyverse)
data(Boston)
boston <- Boston %>% dplyr::select(crim, rm, zn, chas, ptratio, medv)
n_all <- nrow(boston)
set.seed(0)
tr_ind <- sample(n_all, round(n_all/2))
boston_tr <- boston[tr_ind, ]
boston_val <- boston[-tr_ind, ] #validation set
```

```
mod_formula_seq <- c("medv ~ rm",
                     "medv ~ rm + ptratio",
                     "medv ~ rm + ptratio + zn",
                     "medv ~ rm + ptratio + zn + crim")
sapply(mod_formula_seq, function(form){
  mod <- as.formula(form)
  fit <- lm(mod, data = boston_tr)
  pred_medv <- predict(fit, newdata = boston_val)
  mean((boston_val$medv - pred_medv)^2)
})
```

```
##                       medv ~ rm             medv ~ rm + ptratio
##                        48.81990                        42.12556
##        medv ~ rm + ptratio + zn medv ~ rm + ptratio + zn + crim
##                        41.61003                        39.78039
```

Leave-One-Out Cross-Validation (LOOCV)

```
mod_formula_seq <- c("medv ~ rm",
                     "medv ~ rm + ptratio",
                     "medv ~ rm + ptratio + zn",
                     "medv ~ rm + ptratio + zn + crim")
sapply(mod_formula_seq, function(form){
```

```
  mod <- as.formula(form)
  mean(sapply(1:nrow(boston), function(j){
    fit <- lm(mod, data = boston[-j, ])
    pred_medv <- predict(fit, newdata = boston[j, ])
  (boston$medv[j] - pred_medv)^2
  }))
}
)
```

```
##                       medv ~ rm             medv ~ rm + ptratio
##                        44.21666                        37.70066
##        medv ~ rm + ptratio + zn medv ~ rm + ptratio + zn + crim
##                        37.41967                        34.85215
```

```
library(boot)
sapply(mod_formula_seq, function(form){
  mod <- as.formula(form)
  fit <- glm(mod, data = boston)
  cv.glm(boston, fit)$delta[1]
}
)
```

```
##                       medv ~ rm             medv ~ rm + ptratio
##                        44.21666                        37.70066
##        medv ~ rm + ptratio + zn medv ~ rm + ptratio + zn + crim
##                        37.41967                        34.85215
```

k-fold CV for model selection

- Suppose in the `Default` dataset, we are deciding between three models
- model 1: use `rm` to predict `medv`
- model 2: use `rm` and `ptratio` to predict `medv`
- model 3: use `rm`, `ptratio`, and `zn` to predict `medv`
- To choose between these three models using 5-fold CV, we calculate $CV_{(5)}$ for models 1, 2, 3 and choose the model with the smallest $CV_{(5)}$.
- The same idea extends to the case where we need to choose among many models.

```
##K-Fold CV
K <- 5
set.seed(0)
fold_ind <- sample(1:K, n_all, replace = TRUE)
mod_formula_seq <- c("medv ~ rm",
                     "medv ~ rm + ptratio",
                     "medv ~ rm + ptratio + zn")
sapply(mod_formula_seq, function(form){
  mod <- as.formula(form)
  mean(sapply(1:K, function(j){
    fit <- lm(mod, data = boston[fold_ind != j, ])
    pred_medv <- predict(fit, newdata = boston[fold_ind == j, ])
  mean((boston$medv[fold_ind == j] - pred_medv)^2)
  }))
```

```
}
)
```

```
##                medv ~ rm      medv ~ rm + ptratio medv ~ rm + ptratio + zn
##                 43.91820                 37.58662                 37.31531
```

```
library(boot)
set.seed(0)
sapply(mod_formula_seq, function(form){
  mod <- as.formula(form)
  fit <- glm(mod, data = boston)
  cv.glm(boston, fit, K = 5)$delta[1]
}
)
```

```
##                medv ~ rm      medv ~ rm + ptratio medv ~ rm + ptratio + zn
##                 44.12541                 37.74077                 37.73066
```

Use k-fold CV for classification

- The basic idea of CV for classification is the same as that for regression, except the way to calculate $CV_{(k)}$
  - For example, the LOOCV error rate is

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} Err_i\,,$$

  - where $Err_i = I(y_i \neq \hat{y}_i)$

```
library(tidyverse)
library(ISLR)
K <- 5
n_all <- nrow(Default)
fold_ind <- sample(1:K, n_all, replace = TRUE)
mod_formula_seq <- c("default ~ balance",
                     "default ~ balance + income")
sapply(mod_formula_seq, function(form){
  mod <- as.formula(form)
  mean(sapply(1:K, function(j){
    fit <- glm(mod, data = Default[fold_ind != j, ],
               family = "binomial")
    pred_prob <- predict(fit, newdata = Default[fold_ind == j, ], type = "response")
    pred_label <- ifelse(pred_prob > 0.5, "Yes", "No")
  mean(Default$default[fold_ind == j] != pred_label)
  }))
}
)
```

```
##          default ~ balance default ~ balance + income
##                 0.02752060                 0.02621431
```

cv.glm for logistic regression

- Since default choice of the cost function in cv.glm is MSE, we need to define our own cost function to for classification problem
- Things are particularly tricky for logsitic regression because the fit just gives a trained sigmoid function without the decision rule

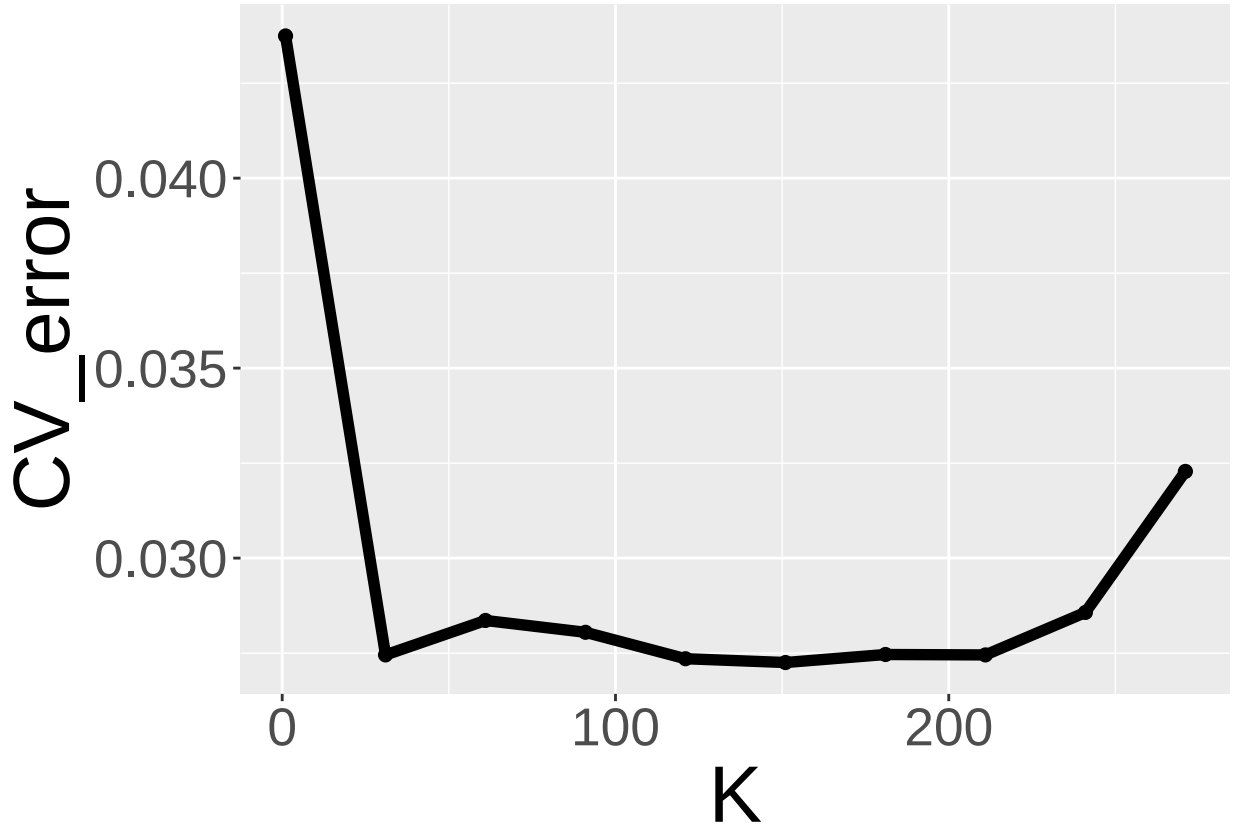
```
K <- 5
n_all <- nrow(Default)
mod_formula_seq <- c("default ~ balance",
                     "default ~ balance + income")
my_cost <- function(r, pi = 0) {
  my_pred <- pi > 0.5
  mean(r != my_pred)
}

# cost <- function(r, pi = 0) mean(abs(r-pi) > 0.5)

sapply(mod_formula_seq, function(form){
  mod <- as.formula(form)
  fit <- glm(mod, data = Default, family = "binomial")
  cv.glm(Default, fit, cost = my_cost, K = 5)$delta[1]
  }
)
```

```
##          default ~ balance default ~ balance + income
##                     0.0277                     0.0263
```

Choosing the K in KNN

```
library(caret)
library(ISLR)
K <- 5
n_all <- nrow(Default)
fold_ind <- sample(1:K, n_all, replace = TRUE)
K_seq <- seq(from = 1, to = 300, by = 30)
CV_error_seq <- sapply(K_seq, function(K_cur){
  mean(sapply(1:K, function(j){
 fit_knn <- knn3(default ~ balance, data = Default[fold_ind != j, ], k = K_cur)
 pred_knn <- predict(fit_knn, newdata = Default[fold_ind == j, ], type = "class")
 mean(pred_knn != Default$default[fold_ind == j])
}))
})
```

```
knn_re <- data.frame(K = K_seq, CV_error = CV_error_seq)
mytheme <- theme(axis.title = element_text(size = 30),
        axis.text = element_text(size = 20),
        legend.text = element_text(size = 20),
        legend.title = element_text(size = 20))
ggplot(knn_re, mapping = aes(x = K, y = CV_error)) +
  geom_point(size = 2) +
  geom_line(size = 2) +
  mytheme
```



Bootstrap

What is "sample with replacement" (the first example)?

- Suppose there is a box that contains a black ball and a red ball
- Draw one ball from the box, put it back, and then draw another ball from the box; repeat this process multiple times (see different outcomes?)

```
set.seed(1)
sample(c("red","black"), size = 2, replace = TRUE)
```

```
## [1] "red"   "black"
```

```
sample(c("red","black"), size = 2, replace = TRUE)
```

```
## [1] "red" "red"
```

```
sample(c("red","black"), size = 2, replace = TRUE)
```

```
## [1] "black" "red"
```

- When we do sample with replacement, it is possible to get one element twice

Estimating the distribution of sample mean using a single sample

```
library(ggplot2)
n <- 200
set.seed(0)
reps <- 1e4
x <- sapply(1:reps, function(j)rnorm(n, mean = 2, sd = 1))
popu_means <- apply(x, 2, mean)
x1 <- x[,1]
x_boot <- sapply(1:reps, function(j)sample(x1, size = n, replace = TRUE))
boot_means <- apply(x_boot, 2, mean)
boot_dat <- rbind(data.frame(mean = popu_means, type = "population"),
                  data.frame(mean = boot_means, type = "bootstrap"))
ggplot(boot_dat) + geom_density(mapping = aes(x = mean, color = type))
```