Lecture 5 - Cross-Validation

Sondre Elstad

2022-06-25

Introduction

Initially, this lecture was set aside to cover loss functions. However, I believe it is much more important to discuss the concepts of model selection and model assessment. One thing is building statistical models. There are a broad range of different models out there. However, machine learning is not about using some buzzword fashionable model with a cool name to mine data. Data mining is a central part of machine learning, but it serves a greater purpose. As we will discuss further in the lecture on deep learning, machine learning is fundamentally concerned about techniques that allows a machine (a computer, say) to learn from experiences. A huge part of learning from experiences is to assess whether actions pursued were the right or the wrong ones. Model selection is the process of finding the appropriate machine learning technique. Model assessment has to do with testing the chosen model on a set of validation or test data, in order to ensure that the chosen model is as optimally tuned as possible.

Validation set approach

Ideally, we would like to separate our data set into three parts: a training set, a validation set and a test set. The training set is used to train the chosen machine learning algorithm. The validation set is for fine-tuning parameters which the chosen model is dependent on. Finally, the test set is for assessing the performance of the chosen machine learning algorithm on previously unseen observations. If we have sufficient data, one might conceive of a distribution of 50% training data, 25% validation data and 25% test data. Alas, being this fortunate is very seldom. Regularly, we have to be satisfied with data of much smaller magnitude. We might have to compromise: separating the data into a training set and a test set and then use some techniques for using the available training data for training and validation purposes. Hopefully, this will be sufficient to obtain satisfying estimates of the test errors.

One of the simplest techniques we can use for this purpose is the so-called validation set approach. We randomly divide the available set of observations into two parts - a training set and a validation set. The fit the model on the training set. Subsequently, the fitted model is used to predict responses for observations in the validation set. This results in a validation set error rate which constitutes an estimate of the test error.

Leave-One-Out CV (LOOCV)

We split the data into a training set and a validation set. However, we do not create two distinct sets of comparable size. Rather, a single observation (x_1, y_1) is used for validation. The remaining observations $\{(x_2, y_2), ..., (x_n, y_n)\}$ is used for training. We fit the model on the $n-1$ training observations and make a prediction $\hat{y}_1$ for the excluded observation, using its value x_1. We did not use (x_1, y_1) in the training of the model and so $MSE_1 = (y_1 - \hat{y}_1)^2$ is an approximately unbiased estimate of the test error. We then exchange observations (x_1, y_1) and (x_2, y_2) and repeat the process. Doing this for all the observations yields a set of n MSEs: $MSE_1, \ldots, MSE_n$. The LOOCV estimate for the test MSE is the average of these n MSEs:

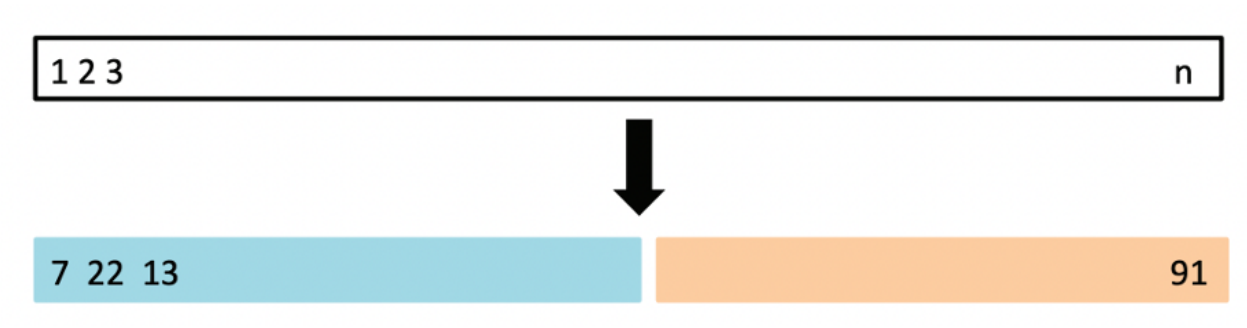

Figure 1: Schematic display of the validation set approach. A set of n observations are randomly split into a training set (blue) and a validation set (beige). The training set is used to fit the model and the validation set is used to evaluate the model. Source: ISL (2017).

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} MSE_i.$$

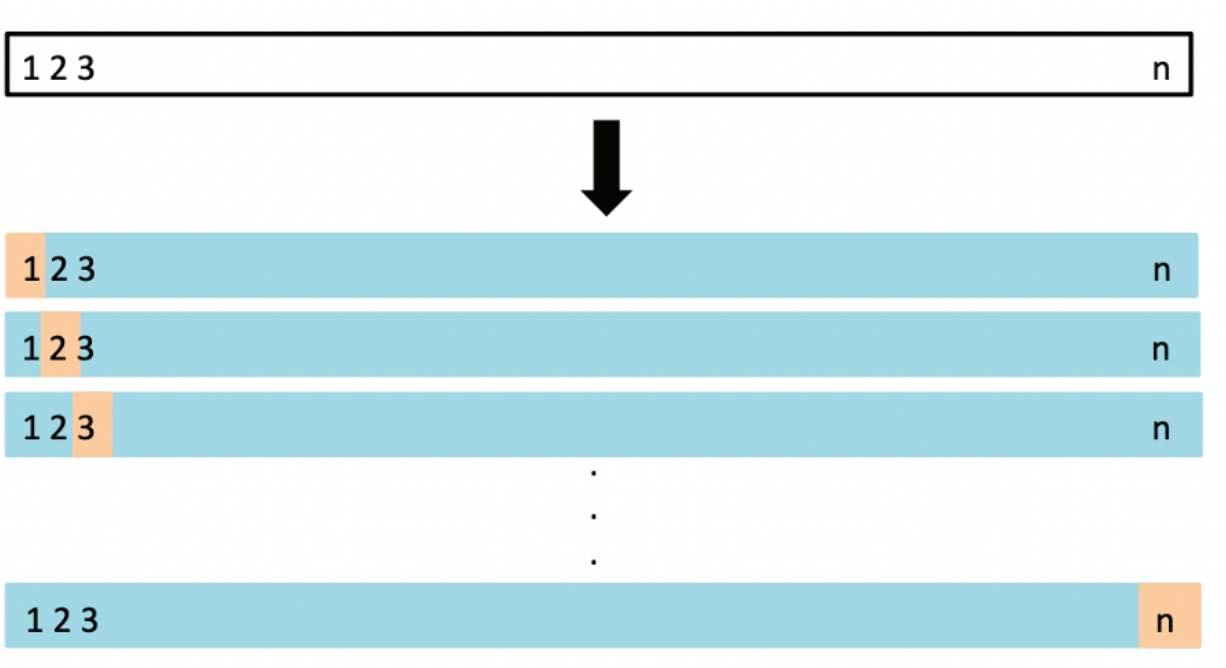

Figure 2: Schematic display of LOOCV. A set of n observations is repeatedly split into a training set (blue) containing every single observation but one. The remaining observation constitutes the validation set. The test error is estimated as the average of the n resulting MSEs. Source: ISL (2017).

k-fold CV

k-fold CV is an intermediate alternative between LOOCV and the validation set approach. Rather than using one and one observation as the validation set and the remaining observations as the training set, we divide the set of observations in to k groups of about the same size. The k-fold CV estimate is computed as

$$CV_{(k)} = \frac{1}{k} \sum_{i=1}^{k} MSE_i.$$

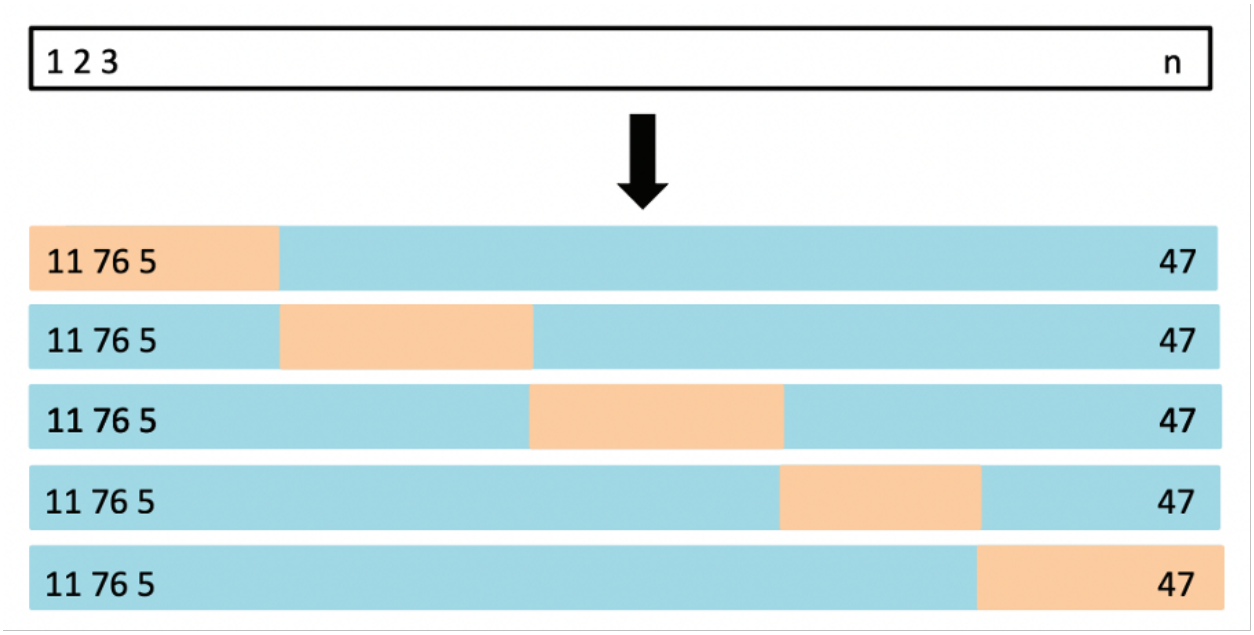

Figure 3: Schematic display of 5-fold CV. The n observations are split into five non-overlapping folds at random. Each fifth is used as validation set in turn (beige) and the remainder is used as training set (blue). Source: ISL (2017).

Recall that LOOCV requires us to train the model n times on $n-1$ observations. This can prove a heavy computational burden. This is especially true for machine learning algorithms who are computationally intensive to begin with (typically very flexible models such as neuron networks). In fact, the argument about flexibility and the bias-varianc tradeoff goes for cross-validation procedures as well. The validation set approach is very simple. It incurs a relatively high bias, but low variance. On the other hand, LOOCV is much more complex. It has low bias but a rather high variance. k-fold CV is an intermediate step. It has lower variance than LOOCV. Since we only change one observation at a time, there is a very high correlation between the different training sets used in LOOCV. On the other hand, since k-fold CV implies exchanging a much larger fraction of the observations from one training set to the next, the correlation between the training sets used in k-fold CV is lower. The mean of many highly correlated quantities has higher variance than the mean of many quantities iwth lower correlation. Finally, we can also apply CV in the classification case:

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

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

R Lab (ISL 2017)

We explore the use of the validation set approach in order to estimate the test error rates that result from fitting various linear models on the Auto data set. Before we begin, we use the set.seed() function in order to set a seed for R's random number generator, so that the reader of this book will obtain precisely the same

results as those shown below. We begin by uing the sample() function to split the set of observations into two halves, by selecting a random subset of 196 observations out of the 392 observations.

```
library(ISLR2)
set.seed(1)
train <- sample(392, 196)
```

We then use the subset option in lm() to fit a linear regression using only the observations corresponding to the training set.

```
lm.fit <- lm(mpg ~ horsepower, data = Auto, subset = train)
```

We now use the predict() function to estimate the response for all 392 observations, and we use the mean() function to calculate the MSE of the 196 observations in the validation set.

```
attach(Auto)
mean((mpg - predict(lm.fit, Auto))[-train]^2)
```

```
## [1] 23.26601
```

We can use the poly() function to estimate the test error for the quadratic and cubic regressions.

```
lm.fit2 <- lm(mpg ~ poly(horsepower, 2), data = Auto,
              subset = train)
mean((mpg - predict(lm.fit2, Auto))[-train]^2)
```

```
## [1] 18.71646
```

```
lm.fit3 <- lm(mpg ~ poly(horsepower, 3), data = Auto,
              subset = train)
mean((mpg - predict(lm.fit3, Auto))[-train]^2)
```

```
## [1] 18.79401
```

The LOOCV estimate can be automatically computed for any generalized linear model using the glm() and cv.glm() functions. In the lab for Chapter 4, we used the glm() function to perform logistic regression by passing in the family = "binomial" argument. But if we use glm() to fit a model without passing in the family argument, then it performs linear regression, just like the lm() function. So for instance,

```
glm.fit <- glm(mpg ~ horsepower, data = Auto)
coef(glm.fit)
```

```
## (Intercept)  horsepower
##  39.9358610  -0.1578447
```

and

```
lm.fit <- lm(mpg ~ horsepower, data = Auto)
coef(lm.fit)
```

```
## (Intercept)  horsepower
##  39.9358610  -0.1578447
```

yield identical linear regression models. The cv.glm() function is part of the boot library.

```
library(boot)
glm.fit <- glm(mpg ~ horsepower, data = Auto)
cv.err <- cv.glm(Auto, glm.fit)
cv.err$delta
```

```
## [1] 24.23151 24.23114
```

The two numbers in the delta vector contain the cross-validation results. We can repeat this procedure for increasingly complex polynomial fits. To automate the process, we use the for() function to initiate a for loop which iteratively fits polynomial regressions for polynomials of order $i = 1$ to $i = 10$, computes the associated cross-validation error, and stores it in the ith element of the vector cv.error.

```
cv.error <- rep(0, 10)
for (i in 1:10) {
  glm.fit <- glm(mpg ~ poly(horsepower, i), data = Auto)
  cv.error[i] <- cv.glm(Auto, glm.fit)$delta[1]
}
cv.error
```

```
##  [1] 24.23151 19.24821 19.33498 19.42443 19.03321 18.97864 18.83305 18.96115
##  [9] 19.06863 19.49093
```

The cv.glm() function can also be used to implement k-fold CV.

```
set.seed(17)
cv.error.10 <- rep(0, 10)
for (i in 1:10) {
  glm.fit <- glm(mpg ~ poly(horsepower, i), data = Auto)
  cv.error.10[i] <- cv.glm(Auto, glm.fit, K=10)$delta[1]
}
cv.error.10
```

```
##  [1] 24.27207 19.26909 19.34805 19.29496 19.03198 18.89781 19.12061 19.14666
##  [9] 18.87013 20.95520
```