CHAPTER 5: PERFORMANCE

CROSS VALIDATION AND OOB

I. Overview

Chapter 4 is about "**Performance**". This is often a concept that seems to be self-evident, and therefore, ignored by people to give further consideration. A model performs well – what does that mean anyway?

For example, let's consider the prediction of a rare disease. By statistics it has been known that only 0.001% of the population of the United States have this disease. The Center of Disease Control (CDC) now hires a data analytics expert to build a model, and it is said that the model can achieve a prediction accuracy as high as 90%. Isn't this a good model? However, it is easy to beat this performance, if we consider a very trivial model that simply predicts all the upcoming cases as negative (no disease). Wouldn't this trivial model achieve a prediction accuracy as high as 99.999%?

Now let's look at another example. Figure 5.1 shows three models to fit the same dataset that has two classes of data points. The first model is a linear model ($f(\boldsymbol{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2$) with a straight line as the decision boundary. Obviously, this model has its inherent limitation such that many data points have to be misclassified with a linear line. To add in some curvature into the decision boundary justified by the nonlinear shape of the two classes' boundaries, some second order terms and an interaction term of the two predictors are introduced to the model ($f(\boldsymbol{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{11} x_1^2 + \beta_{22} x_2^2 + \beta_{12} x_1 x_2$), giving rise to the model shown in the middle panel of Figure 5.1. While this improved model still could not classify the two classes completely, more interaction terms defined on the predictors are introduced into the model ($f(\boldsymbol{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{11} x_1^2 + \beta_{22} x_2^2 + \beta_{12} x_1 x_2 + \beta_{112} x_1^2 x_2 + \beta_{122} x_1 x_2^2 + \cdots$). As shown in the right panel of Figure 5.1, this model can achieve 100% of prediction accuracy.

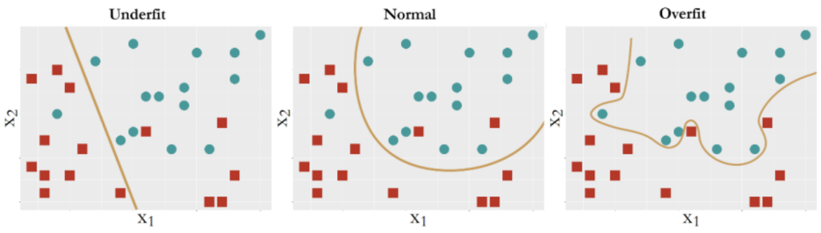

Figure 5.1: Three models to fit a dataset

The three models in Figure 5.1, from left to right, illustrates "**underfit**", "**good fit**", and "**overfit**", respectively. The underfit model, apparently, fails to incorporate something systematical in the dataset to help classify the two classes. The overfit model allows the noises to affect the model. Noises, by definition, only happen by accident. While the model, by definition, is to generalize the constancy of the data rather than its unrepeatable randomness. A dataset could be randomly generated, but the mechanism of generating the randomness is the constancy, as revealed in many phenomena such as Brownian motion. Thus, the model in the middle panel of Figure 5.1 seems to be able to maintain a balance, using the structural constancy in the data to form the model, while resisting the suspicious noises in the data.

A similar study could be conducted on regression problems. As we have mentioned in Chapter 2, the metric R-squared is used to measure the goodness-of-fit of the regression model on training data. Look at the definition of the R-squared,

$$R^2 = 1 - \frac{SSE}{SST}.$$

Here, SST is the total sum of squares, SSE is the residual sum of squares, and it is known that SST-SSE is the explained sum of squares by the model. Thus, R^2 higher the better, meaning more variance in the data could be explained by the model. However, on the other hand, we can see that SST is always fixed, while SSE could only decrease if more variables are put into the model even if these new added variables have no relationship with the outcome variable.

Further, the R-squared is compounded by the variance of predictors as well. As the underlying regression model is

$$Y = \beta X + \epsilon,$$

the variance of Y, $var(Y) = \beta^2 var(X) + var(\epsilon)$. The R-squared takes the form as

$$\text{R-squared}=\frac{\beta^2 var(X)}{\beta^2 var(X)+var(\epsilon)}.$$

Thus, it seems that R-squared is not only impacted by how well X can predict Y, but also by the variance of X as well.

Thus, the drawback of using R-squared is that it doesn't account for model complexity. The adjusted R-squared was developed to provide a remedy for this. Some other criteria such as the **AIC** and **BIC** were also developed which have a good balance between the model fit (just like R-squared) and model complexity (i.e., how many predictors are used in the model).

II. Cross-Validation

II.1 Rationale and Formulation

The examples shown above collectively point out the complexity of defining the performance of a model and the danger of evaluating the performance of a model using training data. To solve this problem, a common strategy is to look at multiple dimensions of the performance of a model, and use **cross-validation** to obtain the performance metrics on a validation dataset that is not used in model training. The ideal situation is that, there is a **training dataset** to train the model and an independent **testing dataset** to validate the model. The testing dataset should not be available when training the model, which is the key for validation purpose. Thus, in training the model with a given dataset, we need to try our best to make sure the model can perform well on the testing dataset. Cross-validation serves this purpose to train the model without accessibility to a testing dataset. The only information that the cross-valiation uses, which is really an assumption, is that the testing dataset and the training dataset are randomly generated by the same distribution. With a given dataset to train the model, the cross-validation techniques mimic the ideal situation, aim to predict the model's success on the unseen testing datasets as its ultimate goal.

II.2 Theory/Method

The first approach, probably the simplest one, is the "hold-out" method. With a given dataset, the hold-out method further divides the given dataset into two parts, a training dataset and a testing dataset. Then, the model is trained on the training dataset. Its performance is evaluated on the testing dataset. Note that, when deciding on the final model that will be used on the testing dataset to obtain its performance, the testing dataset could not be used to guide the model selection on the training stage. In other words, the testing dataset is simply for evaluation only.

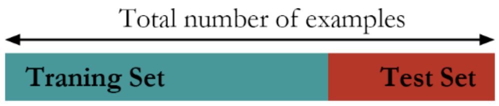

Figure 5.2: The hold-out method

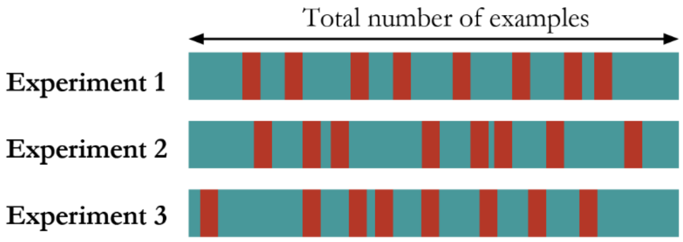

Figure 5.3: The random sampling method

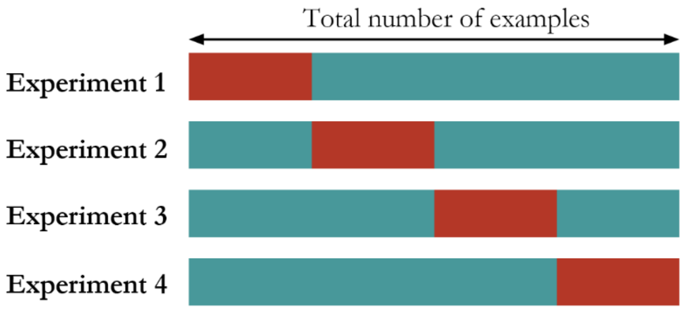

Figure 5.4: The K-fold cross-validation method (here, K=4)

The hold-out method is simple but is criticized for its one-time division of the dataset into two parts, which maybe prone to random errors. Thus, the random sampling method suggests to repeat this division process many times, i.e., as shown in Figure 5.3, the process is repeated 3 times. For each time, the model training and selection only use the training dataset, and the model evaluation only uses the testing dataset. The performance of the model on the three experiments could be reported either in average or in a boxplot that shows both the average performance and its variance.

Somehow like a mix of the random sampling method and the hold-out method, the K-fold cross-validation method suggests to first divide the dataset into K folds, and then, train the model using K-1 folds of the dataset and test the model using the remining fold. This process could be repeated K times. The performance of the model on the experiments could be reported either in average or in a boxplot that shows both the average performance and its variance.

II.3 R Lab

The R lab in this section is built on the script provided in malanor.net[1]. Here, we simulate a simple dataset with one predictor and outcome variable. We use the `ns()` function to simulate the relationship between the two variables, which can generate the B-spline basis matrix for natural cubic splines. The nice merit of using this method is that the relationship between the two variables should be more complex than linear, but the complexity is controlled by the degree of freedom (`df`) parameter, i.e., the larger the `df`, the more complex the relationship. Thus, the complexity of the relationship is quantitatively characterized on a continuum.

```
# Write a nice simulator to generate dataset with one predictor a
nd one outcome
# from a polynomial regression model
seed <- rnorm(1)
set.seed(seed)
gen_data <- function(n, coef, v_noise) {
  eps <- rnorm(n, 0, v_noise)
  x <- sort(runif(n, 0, 100))
  X <- cbind(1,ns(x, df = (length(coef) - 1)))
  y <- as.numeric(X %*% coef + eps)
```

[1] http://www.milanor.net/blog/cross-validation-for-predictive-analytics-using-r/

```
  return(data.frame(x = x, y = y))
}
```

The dataset that is generated by the R code showing above is presented in Figure 5.5, as the scattered data points.

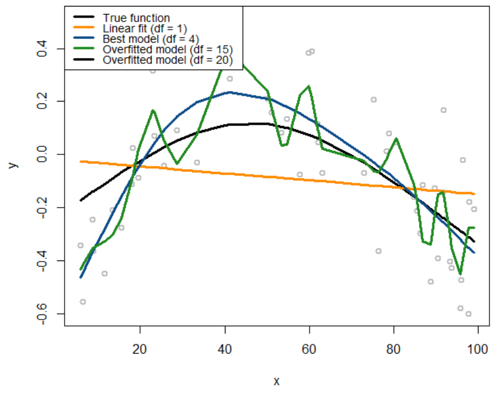

Figure 5.5: The simulated data from a nonlinear regression model with B-spline basis matrix (`df` = 4), and various fitted models with different degree of freedoms

We then fit the data with a variety of models, starting from `df` =1 (corresponds to the linear model) to `df` =20.

```
# Fit the data using different models with different degrees of f
reedom (df)
# install.packages("splines")
require(splines)

## Loading required package: splines

# Simulate one batch of data, and see how different model fits wi
th df from 1 to 50

n_train <- 100
coef <- c(-0.68,0.82,-0.417,0.32,-0.68)
v_noise <- 0.2
n_df <- 20
df <- 1:n_df
tempData <- gen_data(n_train, coef, v_noise)
```

```
x <- tempData[, "x"]
y <- tempData[, "y"]
fit <- apply(t(df), 2, function(degf) lm(y ~ ns(x, df = degf)))


# Plot the data
x <- tempData$x
X <- cbind(1, ns(x, df = (length(coef) - 1)))
y <- tempData$y
plot(y ~ x, col = "gray", lwd = 2)
lines(x, X %*% coef, lwd = 3, col = "black")
lines(x, fitted(fit[[1]]), lwd = 3, col = "darkorange")
lines(x, fitted(fit[[4]]), lwd = 3, col = "dodgerblue4")
# lines(x, fitted(fit[[10]]), lwd = 3, col = "darkorange")
lines(x, fitted(fit[[20]]), lwd = 3, col = "forestgreen")
legend(x = "topleft", legend = c("True function", "Linear fit (df
 = 1)", "Best model (df = 4)", "Overfitted model (df = 15)",
                                 "Overfitted model (df = 20)"), l
wd = rep(3, 4), col = c("black", "darkorange", "dodgerblue4",

                        "forestgreen"), text.width = 32, cex = 0.
85)
```

As shown in Figure 5.5, the linear model obviously underfits the data as it lacks the flexibility to characterize the complexity sufficiently. The model that has `df` =20 overfits the data, evidenced by its complex shape with many change points, up and downs. As we know that the underlying true model is smooth, the model with `df`=20 tries too hard to fit the local patterns that were only induced by noise.

Note that, in this example, we have known that the true `df` is 4, which helps us to recognize the overfitted and underfitted models. In reality we don't have this knowledge, so it is dangerous to keep increasing the model complexity to aggressively pursue the accuracy performance on the training data. To see the danger, let's do another experiment.

First, we use the following R code to generate test data from the same distribution of the training data.

```
# Generate test data from the same model
n_test <- 50
xy_test <- gen_data(n_test, coef, v_noise)
```

Then, we fit the same set of models from linear to `df`=20 using the training dataset. And we compute the prediction errors of these models using the training dataset and testing dataset separately.

```
# Compute the training and test errors for each model
mse <- sapply(fit, function(obj) deviance(obj)/nobs(obj))
pred <- mapply(function(obj, degf) predict(obj, data.frame(x = xy
_test$x)),
                     fit, df)
te <- sapply(as.list(data.frame(pred)), function(y_hat) mean((xy_
test$y - y_hat)^2))
```

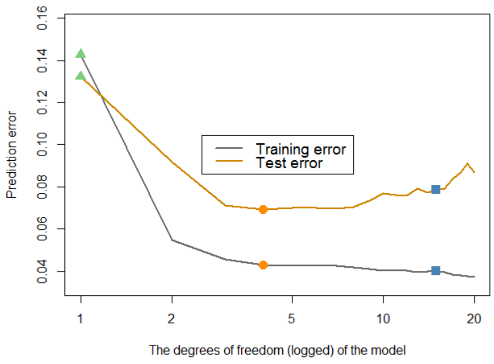

Figure 5.6: Prediction errors of the models (from `df`=0 to `df`=20) on the training dataset and testing data

The following R code is used to draw the Figure 5.6.

```
# Plot the errors
plot(df, mse, type = "l", lwd = 2, col = gray(0.4), ylab = "Predi
ction error",
     xlab = "The degrees of freedom (logged) of the model", ylim
= c(0.9*min(mse), 1.1*max(mse)), log = "x")

lines(df, te, lwd = 2, col = "orange3")

points(df[1], mse[1], col = "palegreen3", pch = 17, cex = 1.5)
points(df[1], te[1], col = "palegreen3", pch = 17, cex = 1.5)
points(df[which.min(te)], mse[which.min(te)], col = "darkorange",
 pch = 16,
       cex = 1.5)
points(df[which.min(te)], te[which.min(te)], col = "darkorange",
pch = 16,
       cex = 1.5)
points(df[15], mse[15], col = "steelblue", pch = 15, cex = 1.5)
points(df[15], te[15], col = "steelblue", pch = 15, cex = 1.5)
legend(x = "center", legend = c("Training error", "Test error"),
lwd = rep(2, 2),
       col = c(gray(0.4), "orange3"), text.width = 0.3, cex = 1.2)
```

As we can see from Figure 5.6, the prediction error on the training dataset keeps decreasing with the increase of the `df`. This is consistent with our theory, and it is important to keep in mind that this triumph of model complexity doesn't really mean what it seems. It only indicates a universal phenomenon that a more complex model can fit the data better. On the other hand, we could observe that the testing error curve shows a U-shape curve, indicating an optimal model could be identified in the examined spectrum of model complexity.

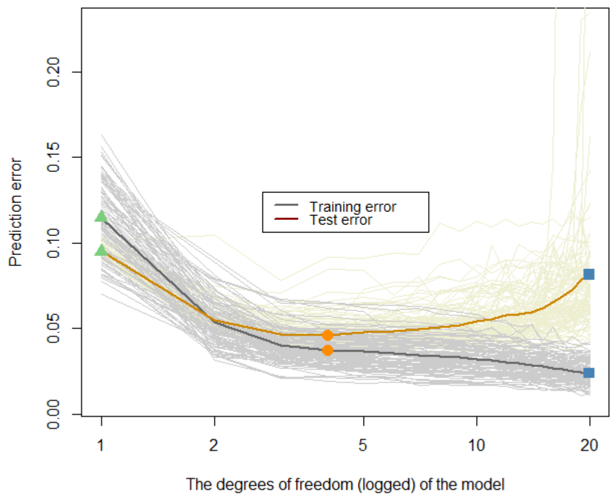

Figure 5.7: Prediction errors of the models (from `df`=0 to `df`=20) on the training dataset and testing data of 100 replications. The two highlighted curves represent the mean curves of the 100 replications of the training and testing error curves, respectively

The following R code repeats this experiment 100 times and present the results in Figure 5.7.

```
# Repeat the above experiments in 100 times
n_rep <- 100
n_train <- 50
coef <- c(-0.68,0.82,-0.417,0.32,-0.68)
v_noise <- 0.2
n_df <- 20
df <- 1:n_df
xy <- res <- list()
xy_test <- gen_data(n_test, coef, v_noise)
for (i in 1:n_rep) {
  xy[[i]] <- gen_data(n_train, coef, v_noise)
  x <- xy[[i]][, "x"]
  y <- xy[[i]][, "y"]
  res[[i]] <- apply(t(df), 2, function(degf) lm(y ~ ns(x, df = de
gf)))
}


# Compute the training and test errors for each model
pred <- list()
mse <- te <- matrix(NA, nrow = n_df, ncol = n_rep)
for (i in 1:n_rep) {
  mse[, i] <- sapply(res[[i]], function(obj) deviance(obj)/nobs(o
bj))
  pred[[i]] <- mapply(function(obj, degf) predict(obj, data.frame
(x = xy_test$x)),
                      res[[i]], df)
  te[, i] <- sapply(as.list(data.frame(pred[[i]])), function(y_ha
t) mean((xy_test$y -

           y_hat)^2))
}

# Compute the average training and test errors
av_mse <- rowMeans(mse)
av_te <- rowMeans(te)

# Plot the errors
plot(df, av_mse, type = "l", lwd = 2, col = gray(0.4), ylab = "Pr
ediction error",
     xlab = "The degrees of freedom (logged) of the model", ylim
= c(0.7*min(mse), 1.4*max(mse)), log = "x")
for (i in 1:n_rep) {
  lines(df, te[, i], col = "lightyellow2")
}
for (i in 1:n_rep) {
  lines(df, mse[, i], col = gray(0.8))
}
lines(df, av_mse, lwd = 2, col = gray(0.4))
lines(df, av_te, lwd = 2, col = "orange3")
points(df[1], av_mse[1], col = "palegreen3", pch = 17, cex = 1.5)
points(df[1], av_te[1], col = "palegreen3", pch = 17, cex = 1.5)
points(df[which.min(av_te)], av_mse[which.min(av_te)], col = "dar
korange", pch = 16,
       cex = 1.5)
points(df[which.min(av_te)], av_te[which.min(av_te)], col = "dark
orange", pch = 16,
       cex = 1.5)
points(df[20], av_mse[20], col = "steelblue", pch = 15, cex = 1.5)
points(df[20], av_te[20], col = "steelblue", pch = 15, cex = 1.5)
legend(x = "center", legend = c("Training error", "Test error"),
lwd = rep(2, 2),
       col = c(gray(0.4), "darkred"), text.width = 0.3, cex = 0.8
5)
```

Next, let's see how well the cross-validation could help to overcome the danger of overfitting. Let's consider the scenario that only the 100 samples are provided to us for both model training and testing. Thus, we use the 10-folder cross-validation on the 100 samples, using the following R code, to train the model.

```
# Cross-validation
set.seed(seed)

n_train <- 100
xy <- gen_data(n_train, coef, v_noise)
x <- xy$x
y <- xy$y

fitted_models <- apply(t(df), 2, function(degf) lm(y ~ ns(x, df =
 degf)))
mse <- sapply(fitted_models, function(obj) deviance(obj)/nobs(ob
j))

n_test <- 10000
xy_test <- gen_data(n_test, coef, v_noise)
pred <- mapply(function(obj, degf) predict(obj, data.frame(x = xy
_test$x)),
               fitted_models, df)
te <- sapply(as.list(data.frame(pred)), function(y_hat) mean((xy_
test$y - y_hat)^2))

n_folds <- 10
folds_i <- sample(rep(1:n_folds, length.out = n_train))
cv_tmp <- matrix(NA, nrow = n_folds, ncol = length(df))
for (k in 1:n_folds) {
  test_i <- which(folds_i == k)
  train_xy <- xy[-test_i, ]
  test_xy <- xy[test_i, ]
  x <- train_xy$x
  y <- train_xy$y
  fitted_models <- apply(t(df), 2, function(degf) lm(y ~ ns(x, df
 = degf)))
  x <- test_xy$x
  y <- test_xy$y
  pred <- mapply(function(obj, degf) predict(obj, data.frame(ns(x,
 df = degf))),
                      fitted_models, df)
  cv_tmp[k, ] <- sapply(as.list(data.frame(pred)), function(y_hat)
 mean((y -

          y_hat)^2))
}
cv <- colMeans(cv_tmp)
```

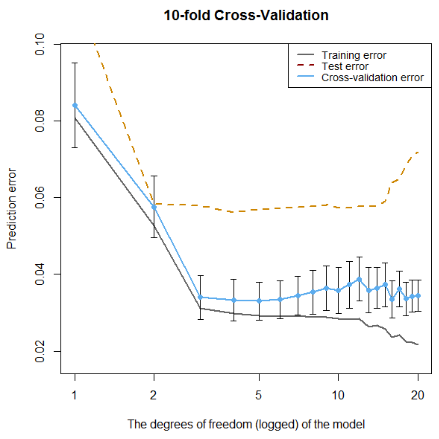

Figure 5.8: Prediction errors of the models (from `df`=0 to `df`=20) on the training dataset without cross-validation, on the training dataset using 10-folder cross-validation, and testing data of 50 samples.

Then we can visualize the result in Figure 5.8. Note that, in Figure 5.8, we overlay the result of the 10-folder cross-validation (based on the 100 samples) with the prediction error on a separate testing dataset (extra 50 samples) to get an idea how close the 10-folder cross-validation can match the ideal case with an extra batch of testing data.

```r
# install.packages("Hmisc")
require(Hmisc)

plot(df, mse, type = "l", lwd = 2, col = gray(0.4), ylab = "Predi
ction error",
     xlab = "The degrees of freedom (logged) of the model", main
= paste0(n_folds,

                  "-fold Cross-Validation"), ylim = c(0.8*min(ms
e), 1.2*max(mse)), log = "x")
lines(df, te, lwd = 2, col = "orange3", lty = 2)
cv_sd <- apply(cv_tmp, 2, sd)/sqrt(n_folds)
errbar(df, cv, cv + cv_sd, cv - cv_sd, add = TRUE, col = "steelbl
ue2", pch = 19,
       lwd = 0.5)
lines(df, cv, lwd = 2, col = "steelblue2")
points(df, cv, col = "steelblue2", pch = 19)
legend(x = "topright", legend = c("Training error", "Test error",
 "Cross-validation error"),
       lty = c(1, 2, 1), lwd = rep(2, 3), col = c(gray(0.4), "dar
kred", "steelblue2"),
       text.width = 0.4, cex = 0.85)
```

As shown in Figure 5.8, the 10-folder cross-validation could generate fair evaluation of the models just like an indepdenent unseen testing dataset. Although its estimation of the error is smaller than the error estimation on the testing dataset, it could capture the change point of model complexity beyond which the model starts to overfit the data. Thus, it could be used to identify a good model that fits the data well, without overfitting.

Now let's apply the idea of 10-folder cross-validation on the AD data, for building a linear regression model with the demographic variables.

```r
#### Dataset of Alzheimer's Disease
#### Objective: prediction of diagnosis
# filename
AD <- read.csv('AD_bl.csv', header = TRUE)
str(AD)
```

```r
# fit a model with demographics only
lm.AD_demo <- lm(MMSCORE ~ AGE + PTGENDER + PTEDUCAT, data = AD)
summary(lm.AD_demo)

n_folds <- 10
folds_i <- sample(rep(1:n_folds, length.out = dim(AD)[1]))
cv_tmp <- matrix(NA, nrow = n_folds, 1)
cv_err <- matrix(NA, nrow = 50*n_folds,2)
for (k in 1:n_folds) {
  test_i <- which(folds_i == k)
  train_xy <- AD[-test_i, ]
  test_xy <- AD[test_i, ]
  y <- test_xy$MMSCORE
  lm.AD_demo <- lm(MMSCORE ~ AGE + PTGENDER + PTEDUCAT, data = tr
ain_xy)
  pred <- predict(lm.AD_demo,test_xy)
  cv_tmp[k] <- mean((y - pred )^2)
  temp <- y - pred
  cv_err[(1+((k-1)*50)):(k*50),1] = rep(k,50)
  cv_err[(1+((k-1)*50)):(k*50),2] <- temp[1:50]
}
```

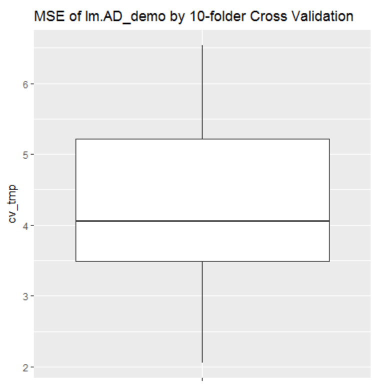

Figure 5.9: Prediction errors of the linear regression model using 10-folder cross-validation

We can use the boxplot to draw the distribution of the prediction errors (evaluated by MSE) collected by the 10-folder cross-validation, shown in Figure 5.9:

```
library(ggplot2)
p <- ggplot(data.frame(cv_tmp),aes(x=factor(""),y=cv_tmp))+geom_b
oxplot()+ xlab("") ## box plot
p <- p + labs(title="MSE of lm.AD_demo by 10-folder Cross Validat
ion")
print(p)
```

Further, while it is not usual in applications to see the prediction errors within the folders, here, it is of interest to present this intermediate result to gain a visual understanding of cross-validation. The R code below draws the boxplots of the prediction errors of the 10 folders, shown in Figure 5.10.

```
# Visualize the distributions of the prediction errors in the fol
ders
cv_err <- data.frame(cv_err)
names(cv_err) = c("Folder","Error")
ggplot(data = cv_err, aes(x = Folder, y = Error)) +
  geom_boxplot(aes(colour=factor(Folder)), fill=NA) +
  geom_point(aes(color = factor(Folder)))
```

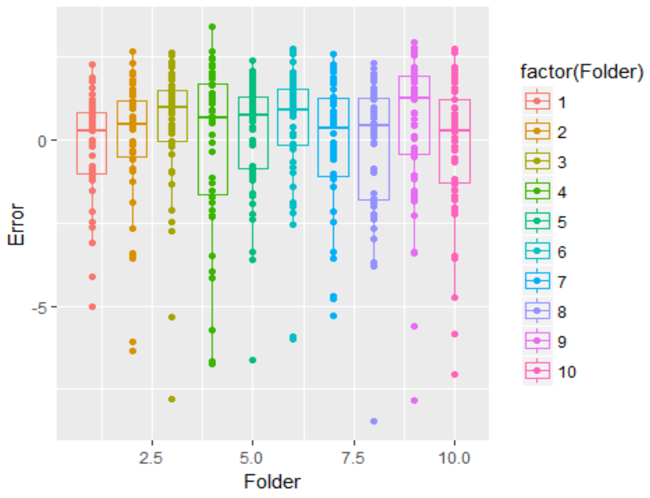

Figure 5.10: Prediction errors of the linear regression model using 10-folder cross-validation; each boxplot corresponds to one folder

II.4 Remarks

More about corss-validation: Usually, there is a relationship between the performance of the model on training dataset and its performance on testing dataset, as shown in Figure 5.11. Note that this relationship is theoretical, but has very high relevance with real applications. In our experimetns, as shown in Figures 5.6 and 5.7, we have seen this relationship. This relationship predicts that, while the performance on the training data will decrease if we increase the model complexity, at a certain point, the gain on performance by increasing model complexity will stop. Beyond this point, the performance would be worse. Thus, a model that has a good performance on the training data and a reasonable complexity is likely to be among the best models that will perform well on the testing data (unseen).

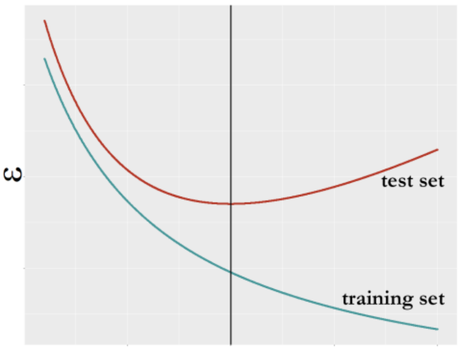

Figure 5.11: A theoretical relationship between the performance of the model on training dataset and its performance on testing dataset

The ROC curve: While cross-validation is useful for estimating the model's performance on unseen testing data, it still needs evaluation metrics to evaluate the model's performance. In some applications such as the rare disease example mentioned earlier in this chapter, how to evaluate the performance of a model itself could be a complex issue.

There have been many performance metrices developed in the literature. An important one, for classification problem, is the ROC curve. As we have seen the limitation of merely using accuracy as the performance metric of a classification model, the ROC has been commonly used as a better metric. The ROC stands for **Receiver Operating Characteristics**. As in a binary classification problem that there are two classes, we often care about accuracies of prediction on both classes. If the classification problem is in a medical application, one class represents disease (positive) while another one represents normal (negative), then we may further name the correct prediction on a positive case as **true positive** (TP) and name the correct prediction on a negative case as **true negative** (TN). Correspondingly, we can define the **false positive** (FP) as incorrect prediction on a positive case and **false negative** (FN) as incorrect prediction on a negative case. This is summered in the following table:

Table 5.1: The confusion matrix

The confusion matrix		**Reality**	
		Positive	Negative
Model Prediction	**Positive**	**True positive (TP)**	**False positive (FP)**
	Negative	**False negative (FN)**	**True negative (TN)**

Now, recall that, in a logistic regression model, before we reach the endpoint of the model that is binary prediction, we obtain the intermediate result $p(\boldsymbol{x}) = \frac{1}{1+e^{-\left(\beta_0+\sum_{i=1}^{p}\beta_i x_i\right)}}$. Then, a cut-off value (e.g., 0.5) is used to classify the cases whose $p(\boldsymbol{x})$ is larger than the cut-off value as one class and otherwise as another class. This means that, for each cut-off value, we can obtain a confusion matrix with different values on the TP and FP. This is shown in Figure 5.12.

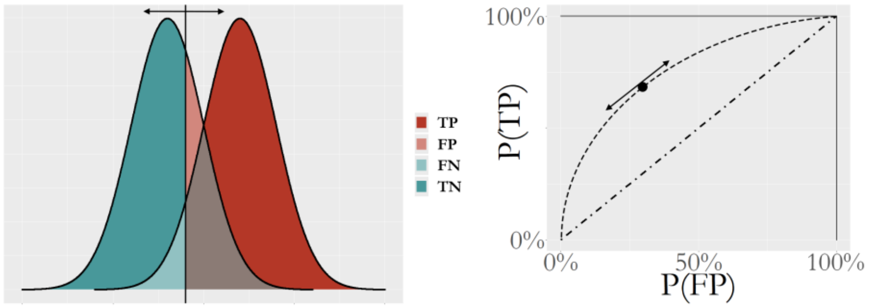

Figure 5.12: For a logistic regression model of two classes, the logistic model can produce the intermediate results $p(\boldsymbol{x})$ for the cases of both classes. (a) shows the distributions of $p(\boldsymbol{x})$ of both classes and a particular cut off value; (b) shows the corresponding confusion matrix; (c) shows the ROC curve that synthesizes all the scenarios of all cut off values

As we can see from Figure 5.12, the ROC curve is a succinct way to synthesizes all the scenarios of all cut-off values. Thus, it provides a more holistic way to evaluate a model (actually, more about to evaluate the potential of a model). A model that lacks potential for prediction will be close to the 45^o line, representing random guess on both classes. A better model will show a ROC curve that is closer to the upper left corner point.

Based on ROC, a metric that is named the **AUC** (the area under the curve) is proposed to summarize the ROC curve of a model. The higher the AUC, the better the model.

In what follows we show how to derive these performance metrics using the logistic regression model. First, let's build a logistic regression model using the AD data as what we have done in Chapter 3.

```r
# ROC and more performance metrics of logistic regression model
# Load the AD dataset
AD <- read.csv('AD_bl.csv', header = TRUE)
str(AD)
```

```
# Split the data into training and testing sets
n = dim(AD)[1]
n.train <- floor(0.8 * n)
idx.train <- sample(n, n.train)
AD.train <- AD[idx.train,]
AD.test <- AD[-idx.train,]

# Automatic selection of the model
logit.AD.full <- glm(DX_bl ~ ., data = AD.train[,c(1:16)], family
 = "binomial")
logit.AD.final <- step(logit.AD.full, direction="both", trace = 0)
summary(logit.AD.final)
```

Then, we can use the function, `confusionMatrix()` from the R package "`e1071`" to derive the performance metrics:

```
# install.packages("e1071")
require(e1071)

require(caret)

# Prediction scores
pred = predict(logit.AD.final, newdata=AD.test,type="response")
confusionMatrix(data=factor(pred>0.5), factor(AD.test[,1]==1))
```

The results are shown in below:

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE    48    7
##      TRUE      7   42
##
##                Accuracy : 0.8654
##                  95% CI : (0.7845, 0.9244)
##     No Information Rate : 0.5288
##     P-Value [Acc > NIR] : 3.201e-13
##
##                   Kappa : 0.7299
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.8727
##             Specificity : 0.8571
##          Pos Pred Value : 0.8727
##          Neg Pred Value : 0.8571
##              Prevalence : 0.5288
##          Detection Rate : 0.4615
##    Detection Prevalence : 0.5288
##       Balanced Accuracy : 0.8649
##
##        'Positive' Class : FALSE
##
```

The ROC curve could be drew using the R Package "ROCR":

```
# Generate the ROC curve using the testing data
# Compute ROC and Precision-Recall curves
require('ROCR')

linear.roc.curve <- performance(prediction(pred, AD.test[,1]),
                                 measure='tpr', x.measure='fpr' )
plot(linear.roc.curve,  lwd = 2, col = "orange3",
     main = "Validation of the logistic model using testing data")
```

The ROC curve is shown in Figure 5.13.

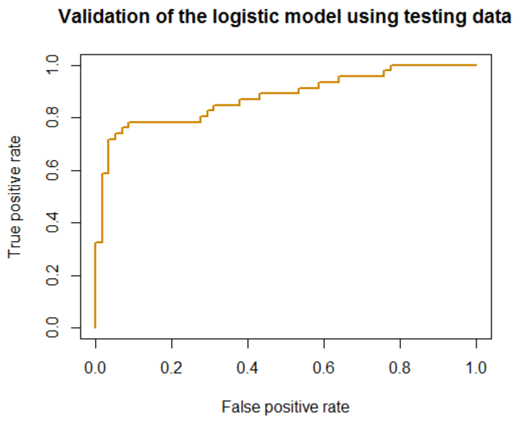

Figure 5.13: ROC curve of the logistic regression model

III. Out-of-bag error in Random Forest

III.1 Rationale and Formulation

The out-of-bag (OOB) error in a random forest model provides a computationally convenient approach to evaluate the model without using a testing dataset, neither a cross-validation procedure. Recall that, for a random forest model with K trees, each tree is built on a bootstrapped dataset from the original training set S. There are totally K bootstrapped datasets, denoted as $B_{1,}B_2, \dots, B_K$.

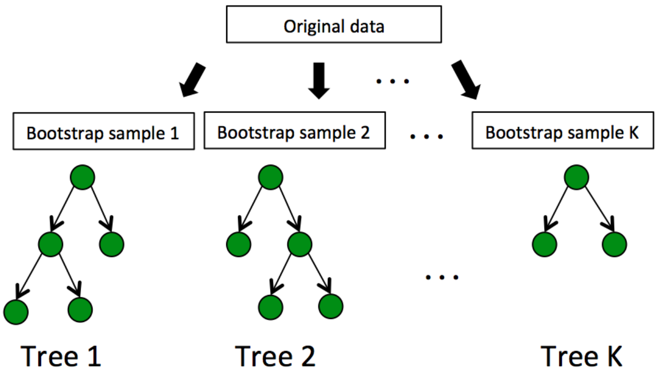

Figure 5.14: The framework of random forest

As the size of each bootstrapped dataset is the same size (denoted as N) as the original training data, and each data point in the bootstrapped dataset is selected independently from other data points, therefore, the probability of a data point from the training data is missing from a bootstrapped dataset is

$$\left(1-\frac{1}{N}\right)^N.$$

When N is sufficiently large, we can have

$$lim_{N\rightarrow\infty}\left(1-\frac{1}{N}\right)^N = e^{-1} \approx 0.37.$$

Therefore, roughly 37% of the data points from S are not contained in any bootstrapped dataset B_i, and thus, not used for training tree i. These excluded data points are referred as the out-of-bag samples for the bootstrapped dataset B_i and tree i. Note that when N is small, the probability of a data point missing from a bootstrapped dataset is smaller, e.g., the probability becomes 0 when $N = 1$, and 1/4 when $N = 2$.

As there are 37% of probability that a data point is not used for training a tree, we can infer that, a data point is not used for training about 37% of the trees. Therefore, for each data point, in theory, there are 37% of trees trained without this data point. These trees can be used to predict on this data point, which can be considered as testing an unseen data point. The out-of-bag error estimation can then be calculated by aggregating the out-of-bag testing error of all the data points. The out-of-bag error can be calculated after random forests are built, and are significantly less computationally than cross-validation. Note that the out-of-bag estimates are calculated by 37% of the trees in the random forest model, therefore, it is expected that the full random forest model with K trees would perform better on any data point than a subset of trees. However, as the performance of the random forest model stabilizes as the number of trees increases, the difference may be small when K is sufficiently large.

Suppose that we have a training dataset of 5 instances (IDs as 1,2,3,4,5). Three trees are built using three bootstrapped datasets, as shown in the Table 5.2.

Table 5.2: Three trees and the corresponding bootstrapped datasets

Bootstrap	Tree
1,1,4,4,5	1
2,3,3,4,4	2
1,2,2,5,5	3

Then, we can estimate the out-of-bag (OOB) errors as shown in Table 5.3 (the true classes of the instances are shown in the top row):

Table 5.3: The out-of-bag (OOB) errors

Tree	Training data	1 (C1)	2 (C2)	3 (C2)	4 (C1)	5 (C2)
1	1,1,4,4,5		C1	C2		
2	2,3,3,4,4	C1				C2
3	1,2,2,5,5			C2	C1	

We can see that, as the data instance (ID = 1) is not used in training Tree 2, we can use Tree 2 to predict on this data instance, and we see that it correctly predict the class as C1. Similarly, Tree 1 is used to predict on data instance (ID=2), and the prediction is wrong. Finally we can see that the overall out-of-bag (OOB) error is 1/6.

III.3 R Lab

We apply the random forest model to the AD dataset and the out-of-bag (OOB) error can be obtained. Random forests are run 50 times. Separately, we use 63/100 of the data for training random forests and use the rest 37/100 data to get testing error (referred to as the validation error), so the proportion of the testing instances similar to the random forests OOB samples. This is repeated 50 times as well. The error rates from both methods are plotted in the boxplots in Figure 5.15. It can be seen that the average error from both methods are similar, but the OOB error seems to have less variance compared to the validation error.

```
library(ggplot2)
require(randomForest)
set.seed(1)

theme_set(theme_gray(base_size = 15))

path <- "../../data/AD_bl.csv"
data <- read.csv(path, header = TRUE)

target_indx <- which(colnames(data) == "DX_bl")
target <- paste0("class_", as.character(data[, target_indx]))
rm_indx <- which(colnames(data) %in% c("DX_bl", "ID", "TOTAL13",
"MMSCORE"))
X <- data
X <- X[, -rm_indx]
```

```
err.mat <- NULL
for (i in 1:50) {
    rf <- randomForest(X, as.factor(target))
    err.mat <- rbind(err.mat, c("OOB_error", mean(rf$err.rate[, "
OOB"])))
}
for (i in 1:50) {
    train.ix <- sample(nrow(X), floor(63 * nrow(X)/100))
    rf <- randomForest(X[train.ix, ], as.factor(target[train.ix]))
    pred.test <- predict(rf, X[-train.ix, ], type = "class")
    err.mat <- rbind(err.mat, c("validation_error", length(which
(pred.test !=
        target[-train.ix]))/length(pred.test)))
}

err.mat <- as.data.frame(err.mat)
colnames(err.mat) <- c("error_type", "error")
err.mat$error = as.numeric(as.character(err.mat$error))
err.mat$error_type <- factor(err.mat$error_type)

ggplot() + geom_boxplot(data = err.mat, aes(x = error_type, y = e
rror)) + geom_point(size = 3)
```

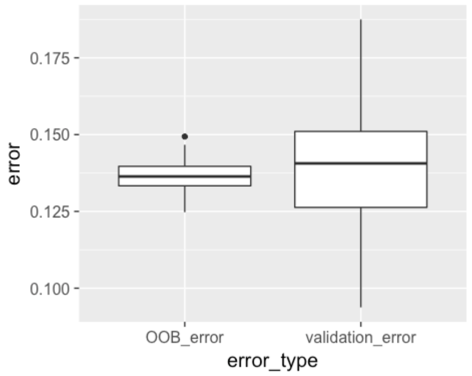

Figure 5.15: Boxplots of the OOB error and validation error

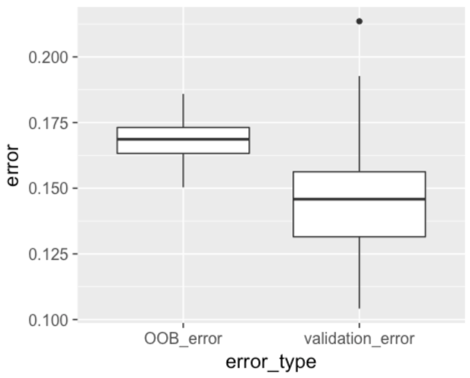

Figure 5.16: Boxplots of the OOB error and validation error

In the previous experiment, the number of trees in the random forest models is the default of 500. Since we expect that, when the number of trees is small, 37% of the trees may not have larger errors than using all the trees. Here, we re-run the previous experiment, but with the number of trees as 50. The OOB errors and validation errors are plotted in Figure 5.16. As expected, the OOB error is clearly larger than the validation error, since only around 37% * 50 trees are used for prediction.

```
err.mat <- NULL
for (i in 1:50) {
    rf <- randomForest(X, as.factor(target), ntree = 50)
    err.mat <- rbind(err.mat, c("OOB_error", mean(rf$err.rate[, "
OOB"])))
}
for (i in 1:50) {
    train.ix <- sample(nrow(X), floor(63 * nrow(X)/100))
    rf <- randomForest(X[train.ix, ], as.factor(target[train.ix]),
 ntree = 50)
    pred.test <- predict(rf, X[-train.ix, ], type = "class")
    err.mat <- rbind(err.mat, c("validation_error", length(which
(pred.test !=
        target[-train.ix]))/length(pred.test)))
}

err.mat <- as.data.frame(err.mat)
```

```
colnames(err.mat) <- c("error_type", "error")
err.mat$error = as.numeric(as.character(err.mat$error))
err.mat$error_type <- factor(err.mat$error_type)

ggplot() + geom_boxplot(data = err.mat, aes(x = error_type, y = e
rror)) + geom_point(size = 3)
```

IV. Exercises

Data analysis

1. Find ten classification datasets from the UCI data repository or R datasets. Using these datasets, conduct experiments to see if the cross-validation method on training data can provide an approximation of the testing error on a testing data, as shown in Figure5.8.
2. Using the datasets you picked up in 1, use cross-validation to select the best logistic regression model, the best decision tree model, and the best random forest model. Compare the models.
3. Using these datasets, build random forest models and compare the OOB error rates from the random forest models with 10-folder cross-validation error.

Programming

In the book[1] by Prof. Cosma Rohilla Shalizi, an interesting experiment is proposed to show another disadvantage of the concept R-squared. Three simple datasets are simulated via:

1. Simulate 100 data points of predictor X from a uniform distribution $Unif(0,1)$; then, simulate 100 corresponding values of response variable Y from $N\left(\sqrt{X}, 0.05^2\right)$.

[1] *Shalizi, C.R. Advanced data analysis from an elementary point of view. Book Manuscript: http://www.stat.cmu.edu/~cshalizi/ADAfaEPoV/ADAfaEPoV.pdf.*

2. Simulate 100 data points of predictor X from $N(0.5, 0.1^2)$; then, simulate 100 corresponding values of response variable Y from $N(\sqrt{X}, 0.05^2)$.
3. Simulate 100 data points of predictor X from a uniform distribution $Unif(2,3)$; then, simulate 100 corresponding values of response variable Y from $N(\sqrt{X}, 0.05^2)$.

Built three regression models on the three datasets. Comment on the R-squareds of the three fitted models (which aim to fit the same regression model anyway).