Linear Regression

Arnab Maity - Modified by Justin Post

Packages used in this set of notes:

library(MASS)  
library(klaR)  
library(tufte)  
library(tidyverse)  
library(lubridate)  
library(caret)  
library(rsample)  
library(ISLR2)  
library(knitr)  
library(AppliedPredictiveModeling)  
library(kableExtra)  
library(robustbase)

# Big Picture

For now, we stick to the regression task *where we have a quantitative response*. We need to select a model form to work with. We’ll start with a basic parametric model - *a model with a stronger structural form* - called the (Multiple) Linear Regression model.

The model is much simpler compared to other modern techniques; however, such models are still very useful in developing new methods. In fact, many flexible nonparametric models can be thought of generalizations of linear regression model. We could spend an entire course on this topic if we wanted to!

Recall that we have two major goals:

* Inference - *determine which predictors are important for the model and quantifying their effects and relationships*
* Prediction - *performing well at predicting responses for observations the data was not trained on*

We’ll see that most of the models we look at in this section can be used for either of these tasks!

# Introduction to the Linear Regression Model

We already investigated the simple linear regression model. This is a model where we have a single predictor being used to model the response.

Let’s reintroduce our data on bike sharing.

bike\_share <- read\_csv("https://www4.stat.ncsu.edu/online/datasets/SeoulBikeData.csv",  
 local = locale(encoding = "latin1"))

Rows: 8760 Columns: 14  
── Column specification ────────────────────────────────────────────────────────  
Delimiter: ","  
chr (4): Date, Seasons, Holiday, Functioning Day  
dbl (10): Rented Bike Count, Hour, Temperature(°C), Humidity(%), Wind speed ...  
  
ℹ Use `spec()` to retrieve the full column specification for this data.  
ℹ Specify the column types or set `show\_col\_types = FALSE` to quiet this message.

bike\_share |>   
 select(`Rented Bike Count`, everything())

# A tibble: 8,760 × 14  
 `Rented Bike Count` Date Hour `Temperature(°C)` `Humidity(%)`  
 <dbl> <chr> <dbl> <dbl> <dbl>  
 1 254 01/12/2017 0 -5.2 37  
 2 204 01/12/2017 1 -5.5 38  
 3 173 01/12/2017 2 -6 39  
 4 107 01/12/2017 3 -6.2 40  
 5 78 01/12/2017 4 -6 36  
 6 100 01/12/2017 5 -6.4 37  
 7 181 01/12/2017 6 -6.6 35  
 8 460 01/12/2017 7 -7.4 38  
 9 930 01/12/2017 8 -7.6 37  
10 490 01/12/2017 9 -6.5 27  
# ℹ 8,750 more rows  
# ℹ 9 more variables: `Wind speed (m/s)` <dbl>, `Visibility (10m)` <dbl>,  
# `Dew point temperature(°C)` <dbl>, `Solar Radiation (MJ/m2)` <dbl>,  
# `Rainfall(mm)` <dbl>, `Snowfall (cm)` <dbl>, Seasons <chr>, Holiday <chr>,  
# `Functioning Day` <chr>

As the variable names are non-standard in R, let’s quickly modify them to make our lives easier (and have better consistency). We’ll also make some variables factors, which are special character variables that only take on a few values (or levels).

bike\_share <- bike\_share |>  
 rename("date" = "Date",  
 "rented\_bike\_count" = `Rented Bike Count`,  
 "hour" = "Hour",  
 "temperature" = `Temperature(°C)`,  
 "humidity" = `Humidity(%)`,  
 "wind\_speed" = `Wind speed (m/s)`,  
 "visibility" = `Visibility (10m)`,  
 "dew\_point\_temperature" = `Dew point temperature(°C)`,  
 "solar\_radiation" = `Solar Radiation (MJ/m2)`,  
 "rainfall" = `Rainfall(mm)`,  
 "snowfall" = `Snowfall (cm)`,  
 "seasons" = "Seasons",  
 "holiday" = "Holiday",  
 "functioning\_day" = "Functioning Day"   
 ) |>  
 mutate(date = dmy(date), #convert the date variable from character  
 seasons = factor(seasons),  
 holiday = factor(holiday),  
 functioning\_day = factor(functioning\_day))

Ok, let’s graph the relationship the SLR model fits between rented\_bike\_count and temperature.

bike\_share |>  
 ggplot(aes(x = temperature, y = rented\_bike\_count)) +  
 geom\_point(size = 0.5) +  
 geom\_smooth(method = "lm")

|  |
| --- |
| Scatterplot with fitted SLR model overlayed |

The equation for the model is fit in the software and given here.

SLR\_fit <- lm(rented\_bike\_count ~ temperature, data = bike\_share)  
summary(SLR\_fit)$coefficients

Estimate Std. Error t value Pr(>|t|)  
(Intercept) 329.9525 8.5410613 38.63132 1.288037e-301  
temperature 29.0811 0.4861734 59.81631 0.000000e+00

Clearly, in real data, we often have relationships that are non-linear. However, sometimes these relationships can be *locally* linear. In other cases, even if the original relationship is not linear, one may transform the response and predictors (e.g., using a -transform) to get approximate linearity.

## Model Must Be **Fit**

Once we’ve determined our model structure, we must **fit** or **estimate** our model. This is generally done by minimizing some criteria.

*The common criterion for the multiple linear regression model is least squares (equivalent to maximum likelihood with normal errors described shortly).*

## Conducting Inference Using the Model

If we want to do statistical tests, we usually need to make assumptions about the error terms of our linear regression model. The assumptions made to do inference are also dependent on how we fit the model.

*In the MLR setting, we usually assume our errors iid by a normal distribution with constant variance. These assumptions require us to investigate model diagnostics in order to understand the validity of the assumptions.*

## Using the Model to Predict

Once the model is fit, we often want to predict. This can be done by plugging in values of our predictor(s) we are interested in. The estimated values is called .

*In the SLR model we have*

Our plan will be to discuss the multiple linear regression model in detail and then look at extensions to the model such as the LASSO, Ridge Regression, Principle Component Regression, etc.

# Multiple Linear Regression (MLR)

A linear regression model has the form

where

* is a quantitative response *(rented\_bike\_count)*
* are predictor variables *(temperature, hour, seasons, etc.)*
* is unobserved random error
* is the *intercept* *(expected bike count when all predictors are 0)*
* Coefficients are ‘slope’ terms associated with predictor
  + Value of indicates the strength, and direction, of the *linear* relationship between and
  + *is the expected change in the response for a unit change in , holding all other predictors constant*
  + *If interactions or quadratics are included, this interpretation changes!*

MLR\_fit <- lm(rented\_bike\_count ~ temperature + hour + wind\_speed, data = bike\_share)  
summary(MLR\_fit)$coefficients

Estimate Std. Error t value Pr(>|t|)  
(Intercept) -48.71548 13.2239366 -3.683887 2.310891e-04  
temperature 26.91385 0.4473949 60.156813 0.000000e+00  
hour 31.30939 0.8048948 38.898735 1.862006e-305  
wind\_speed 26.97563 5.3387085 5.052838 4.440897e-07

## Assumptions on the Errors

### Mean Zero

Typically, we assume that the errors have mean zero, that is . Thus we can write the mean response as

We call the model described above {} because is {}, that is, linear in .

We can have nonlinear terms of , but the model would be still a linear model!

* *For example, the so-called polynomial regression,*
* Fractional polynomial regression (introduced by Royston & Sauerbrei in their 2004 paper, *A new approach to modelling interactions between treatment and continuous covariates in clinical trials by using fractional polynomials.* in Stat Med)
* *where the powers are chosen from*
* Models with interaction such as

A non-linear model, in contrast, is a model such as

Hopefully, you can see that linear models do not just capture linear effects of , they can capture nonlinear functions of as well! The figurebelow shows a few examples functions that can be captured by appropriate linear models.

|  |
| --- |
| Examples of functions that can be captured by appropriate linear models. |

### Common Distributional Assumptions

A normality assumption on the errors, enable us to perform statistical inference on the coefficients such as:

* Constructing confidence intervals for : a set/range of values which contain the “true” value of with high probability (in repeated sampling). This can be investigated by a -statistic based confidence interval.
* Perform hypothesis tests to determine whether the -the predictor has any linear association with , vs. . This can be investigated using a -test.
* Perform hypothesis tests to determine whether the *any* of the predictors has any linear association with , vs.  at least on is non-zero. We can use -test to answer this question.

Although not the only way to make the above inferences, our common assumption on the distribution of the errors is

Conceptually, this idea can be visualized in the simple linear regression model by the graph below.

|  |
| --- |
| At each value of the predictor, we assume the values of the response variable are normally distributed about the line. Figure from physicsforums.com |

## Fitting a Linear Model

We denote the estimated coefficients as and the estimated response as .

How we determine these estimated coefficients depends! Let’s go through common methods for fitting the model.

### Least Squares

A common estimation procedure for the regression coefficients is the *least squares* technique.

* The difference between the observed and predicted values are called *residuals*,
* We define the *residual sum of squares*, also known as *sum-of-squared errors (SSE)* as

We see that MSE is simply RSS divided by the sample size! (This could be found over the training set or a test set.)

The *ordinary least squares (OLS)* procedure estimates by minimizing the sum-of-squares

with respect to ’s (resulting estimates are called the OLS estimates).

For the *simple linear regression* model the solutions are easy to derive with calculus.

* For the general regression model with predictors, writing closed form expression is easier in matrix from. We can convert the original regression model in matrix form as
* where
* is the column vector of responses
* is the column vector of regression coefficients
* is the column vector of errors.
* is matrix
  + We call the *model matrix* or *design matrix*
  + The first column of has all elements equal to (corresponding to the intercept)
  + For the remaining part of , each row corresponds to an unit/individual and each column corresponds to a predictor.
    - For the special case of simple linear regression with one predictor , the model matrix is of size ,
    - In general, we have

For our bike count data, we can write out some of these terms for clarity. Let’s just work with the first 10 observations for brevity.

bike\_share\_first\_ten <- bike\_share[1:10, ]  
y <- bike\_share\_first\_ten$rented\_bike\_count  
X <- bike\_share\_first\_ten |>  
 mutate(intercept = rep(1, 10)) |>  
 select(intercept, temperature, hour, wind\_speed) |>  
 as.matrix()  
y

[1] 254 204 173 107 78 100 181 460 930 490

X

intercept temperature hour wind\_speed  
 [1,] 1 -5.2 0 2.2  
 [2,] 1 -5.5 1 0.8  
 [3,] 1 -6.0 2 1.0  
 [4,] 1 -6.2 3 0.9  
 [5,] 1 -6.0 4 2.3  
 [6,] 1 -6.4 5 1.5  
 [7,] 1 -6.6 6 1.3  
 [8,] 1 -7.4 7 0.9  
 [9,] 1 -7.6 8 1.1  
[10,] 1 -6.5 9 0.5

A *unique* minimizer of the sum-of-squares exists under certain conditions!

We can find this in software easily.

solve(t(X)%\*%X)%\*%t(X)%\*%y

[,1]  
intercept -1010.802886  
temperature -208.213746  
hour 6.925921  
wind\_speed -34.191124

Which is the same as the basic lm() fit!

MLR\_first\_ten <- lm(rented\_bike\_count ~ temperature + hour + wind\_speed,  
 data = bike\_share\_first\_ten)  
MLR\_first\_ten$coefficients

(Intercept) temperature hour wind\_speed   
-1010.802886 -208.213746 6.925921 -34.191124

#### Notes on Matrix Requirements

The estimator above depends on the term , that is, the inverse of . Such an inverse exists only if  *has full column-rank*. Equivalently, must have the following two properties:

(C1) The sample size is larger than the number of regression coefficients in the model, .

(C2) None of the columns of can be written as a weighted sum (called a *linear combination*) of the remaining columns.

If violates either of these conditions, then a *unique* least squares estimator does not exist. If violates (C2) but not (C1), then we can replace by a *generalized inverse*[[1]](#footnote-37), but there will be many estimators that minimize the sum-of-squares. Interpreting them will be difficult in general. In practice, even if the predictors are not perfectly correlated, their correlation can be high enough to cause numerical instability. This issue is known as *multicollinearity* among predictors. We can avoid this issue by removing the collinear predictors from the model.

If violates (C1) then (C2) is automatically violated.[[2]](#footnote-38) In that case, one can take a few steps described below.

* We can *remove highly correlated predictors* to reduce the overall number of predictors.
* Use *variance inflation factor (VIF)* – we will learn it shortly – to diagnose multicollinearity. VIF tells us how correlated each predictor is with the remaining predictors.
* Apply *dimension reduction techniques*, such as principal component analysis (PCA) or partial least squares (PLS).
* Apply *shrinkage methods*, such as LASSO regression, to reduce small regression coefficients to zero.

### Fitting Via Maximum Likelihood (Optional)

If you are familiar with maximum likelihood, the solutions obtained from the least squares optimization are the same as the solutions from the model that assumes our iid Normal errors with constant variance.

Consider the SLR model with Normal errors. The likelihood is given by

We want to find the values of , , and that maximize this function. We can interpret these values as the ‘most likely values of the parameters to have produced the data we saw.’

Rather than optimize the likelihood directly, we usually optimize the log-likelihood (natural log) given by

We can take the derivatives with respect to each of our parameters, set the resulting equations equal to zero and solve simultaneously. But wait! Notice that the maximizing the second term over and is equivalent to minimizing our sum of squared errors (i.e. least squares)!

Therefore, we get the same coefficient estimates for our intercept and slope terms whether we use least squares or maximum likelihood!

### Other Methods for Fitting the Regression Model

A drawback of least squares (and, hence, maximum likelihood with Normally distributed errors)is that it is susceptible to influential points. That is, points that have an unduly high impact on the regression process!

The figure below shows an example of an outlier and its impact of the regression fit.

|  |
| --- |
| Example of an outlier (red point). Shown are two regression lines: before (red dashed) and after (blue solid) removing the outlier. |

We could simply remove the outlier from the data set but that must be done carefully.

We can also use a different minimization criterion that is more robust to influential points.

* We can use the *least absolute deviation (LAD)* criterion,

or

* *Huber function*, which uses squared residuals when their values are small, but uses absolute value for large residuals (above a certain cutoff).

|  |
| --- |
| Visual of the loss associated with squared error loss and huber loss. Source: Wikipedia |

In general, *robust regression* methods are often of interest if data are prone to large outliers or have a heavy tailed distribution.

#uses the robustbase package  
mlr\_ae\_fit <- lmrob.lar(x = as.matrix(bike\_share |>   
 mutate(intercept = rep(1, nrow(bike\_share))) |>  
 select(intercept, temperature, wind\_speed)),  
 y = bike\_share$rented\_bike\_count)  
mlr\_ae\_fit$coefficients

[1] 160.78571 26.55844 72.24026

Compare with the MLR fit:

mlr\_ls\_fit <- lm(rented\_bike\_count ~ temperature + wind\_speed, data = bike\_share)  
mlr\_ls\_fit$coefficients

(Intercept) temperature wind\_speed   
 175.22135 29.35675 87.64519

## Implementation in R

We can use the lm() function in base R to fit linear models. Let us revisit the Boston data in the ISLR2 package. We start by fit a simple linear regression model with medv as response and lstat as the predictor. The model matrix for this regression should have two columns: the first column with 1 as each element, and the second column would contain lstat values. If needed, we can use the model.matrix() function to create manually.

model\_mat <- model.matrix( ~ lstat, data = Boston)  
head(model\_mat)

(Intercept) lstat  
1 1 4.98  
2 1 9.14  
3 1 4.03  
4 1 2.94  
5 1 5.33  
6 1 5.21

The first argument of model.matrix() is the formula “~ lstat”. Note the left hand side of the formula is empty since does not depend on the response. The right hand side contains the regression formula. Note that the intercept is automatically included[[3]](#footnote-50) – we do not need to write “~ 1 + lstat.

We can use the lm() function in base R to fit a linear model. We can use the formula interface directly if we want, as shown below.[[4]](#footnote-51)

simple\_ols <- lm(medv ~ lstat,  
 data = Boston)  
simple\_ols$coefficients

(Intercept) lstat   
 34.5538409 -0.9500494

|  |
| --- |
| Least squares fit to the simple linear regression model with medv as response and lstat as the predictor. |

Figure shows the least squares fit to the Boston data. We can see the estimate intercept, , and the slope . The estimated slope the rate of change in for each unit increase in . In other words, every unit in crease in lstat (lower status of the population in percent), the expected value of medv (median value of owner-occupied homes) decreases by 0.95 units. Examining Figure , there is some evidence of nonlinear effect of lstat for smaller and larger values.

To fit multiple linear regression with more than one predictors, we simply need to include the predictors in the formula. For example, we can fit a quadratic term of lstat as well.[[5]](#footnote-55) Figure shows the fitted regression line.

quad\_ols <- lm(medv ~ lstat + I(lstat^2),  
 data = Boston)  
quad\_ols$coefficients

(Intercept) lstat I(lstat^2)   
42.86200733 -2.33282110 0.04354689

|  |
| --- |
| Least squares fit to the simple linear regression model with medv as response and lstat as the predictor. |

We can fit interaction terms using the X1\*X2 notation. Thus the fomula Y ~ X1\*X2 will include *main effects* of and , and the *two-way interaction effect* in the model. For example, we can fit a model with lstat, age and their interaction as follows.[[6]](#footnote-59)

int\_ols <- lm(medv ~ lstat \* age,  
 data = Boston)  
int\_ols$coefficients

(Intercept) lstat age lstat:age   
36.0885359346 -1.3921168406 -0.0007208595 0.0041559518

We usually *retain all lower-order terms corresponding to an interaction* in the model, that is, if the model has the term , we also retain terms and . In presence of an interaction term, the effect of a predictor on response is not constant. The expected change in the response due to one unit increase in the predictor also depends on the other predictor involved in the interaction. Specifically, consider the model

|  |
| --- |
| Contour plots of fitted surface in regression without (top) and with (bottom) a two-way interaction term. |

Then for a fixed value of , one unit increase in corresponds to amount change in , not just . Thus the fitted surface will have some curvature. Figure shows fitted surface with and without a two-way interaction effect. In the example above, one unit increase in lstat corresponds to a change of in medv on average. On the other hand, one unit increase in age corresponds to a change of in medv on average. We can interpret the estimated interaction term as the effect of age on the impact of lstat on medv (and vise-versa).

# Standard errors

The estimated coefficients, and thus the fitted regression line are random quantities since they change from sample to sample. Thus we need a way to quantify the variability associated with the estimates. To this end, we require additional assumptions on the error terms . For simplicity, we will use the following set of assumptions:[[7]](#footnote-65)

* The errors are independently and identically distributed as .
* Errors are independent of the covariates

The assumptions above imply that

One way to quantify variability associated with estimation of ’s is to compute the *standard error (SE)* of the estimates, defined as,

For a general multiple linear regression model with model matrix , the standard error of can be computed as[[8]](#footnote-66)

A special case is the simple linear regression, where the standard errors, given fixed values of are

Examining the expression of standard errors of and shows that SE will be smallest if the denominator is maximized. Thus, is smallest if the predictor values, ’s, are more spread out from their center.

Note that the standard error expressions depend on the error variance , which is an unknown quantity. We can estimate , and , using the residual sum of squares. We then plug-in in place of in the expressions of standard errors, to obtain *estimated standard errors*, . For simplicity of presentation, we will still denote the estimated standard error by .

The denominator in the expression of merits some discussion. We can view this number as “sample size - number of parameters in the mean function”. In simple linear regression we have two parameters in mean function, and thus we have the term . In general, with predictors, the total number of parameters in the mean function is (since we need to include the intercept). Thus we use as the denominator.

We can see the standard errors in R using the summary() function.

summary(simple\_ols)

Call:  
lm(formula = medv ~ lstat, data = Boston)  
  
Residuals:  
 Min 1Q Median 3Q Max   
-15.168 -3.990 -1.318 2.034 24.500   
  
Coefficients:  
 Estimate Std. Error t value Pr(>|t|)   
(Intercept) 34.55384 0.56263 61.41 <2e-16 \*\*\*  
lstat -0.95005 0.03873 -24.53 <2e-16 \*\*\*  
---  
Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
  
Residual standard error: 6.216 on 504 degrees of freedom  
Multiple R-squared: 0.5441, Adjusted R-squared: 0.5432   
F-statistic: 601.6 on 1 and 504 DF, p-value: < 2.2e-16

The first column of the output above are the estimates that we have discussed before. The second column gives the standard errors of the estimates.

Alternatively, we can directly use the following code:[[9]](#footnote-67)

se <- sqrt(diag(vcov(simple\_ols)))  
se

(Intercept) lstat   
 0.56262735 0.03873342

The last part of the summary output above shows Residual standard error: 6.216. Thus in this case . We can use the sigma() function to directly obtain this value.

sigma(simple\_ols)

[1] 6.21576

# Inference

The standard errors, along with normality assumption on the errors, further enable us to perform statistical inference on the coefficients:

* Construct confidence intervals of : a set/range of values which contain the “true” value of with high probability.[[10]](#footnote-69) This can be investigated by a -statistic based confidence interval.
* Perform hypothesis tests to determine whether the -the predictor has any linear association with , vs. . This can be investigated using a -test.
* Perform hypothesis tests to determine whether the *any* of the predictors has any linear association with , vs.  at least on is non-zero. We can use -test to answer this question.

We discuss each of the items below.

## Confidence interval

Without going into mathematical details, we can obtain a confidence interval for using the standard errors. Again, the number is the same as we see in the estimator . This number is called the *degrees of freedom* of the -distribution used above.

|  |
| --- |
| t PDF and quantiles. The shaded region has area p, and the x-axis value corresponding to the solid vertical line represents the (1-p)-quantile. In this example, we have p = 0.05, and the vertical line represents the upper 0.95-quantile. |

In R, we can use the function confint() to obtain *individual* confidence intervals for the regression coefficients.

## 95\% confidence intervals  
ci <- confint(simple\_ols, level = 0.95)  
ci

2.5 % 97.5 %  
(Intercept) 33.448457 35.6592247  
lstat -1.026148 -0.8739505

We can interpret the intervals for intercept by saying that when lstat = 0, on average medv is estimated to be between 33.45 and 35.66 with confidence. We can interpret the interval for the slope as follows: we estimate with confidence that medv on average decreases between 1.03 and 0.87 for 1 unit increase in lstat.[[11]](#footnote-73)

## test

We can also perform hypothesis tests on the regression coeffieients. If our main interest is in testing the association between and , we test for

Note that implies that is not in the model, and thus not associated with . We can use the -statistic to perform the test:[[12]](#footnote-75) The test statistic measures how far away the estimated value of is from zero compared to the variability of the estimate measured by . We expect the statistic to have a distribution *if is true*. Then we reject is the observed value of is very large or very small compared to what we expect from a distribution. Equivalently, we can compute the p-value of this test as

|  |
| --- |
| Two-tailed p-value for a t-test. |

where is the CDF of the distribution evaluated at . We reject if the p-value if smaller than , where we set to be a small value (usually set to ). The quantity is called the type I error of the test (probability of rejecting when it should not be rejected).

In R, we can use the summary() function to obtain the test results.

summary(simple\_ols)

Call:  
lm(formula = medv ~ lstat, data = Boston)  
  
Residuals:  
 Min 1Q Median 3Q Max   
-15.168 -3.990 -1.318 2.034 24.500   
  
Coefficients:  
 Estimate Std. Error t value Pr(>|t|)   
(Intercept) 34.55384 0.56263 61.41 <2e-16 \*\*\*  
lstat -0.95005 0.03873 -24.53 <2e-16 \*\*\*  
---  
Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
  
Residual standard error: 6.216 on 504 degrees of freedom  
Multiple R-squared: 0.5441, Adjusted R-squared: 0.5432   
F-statistic: 601.6 on 1 and 504 DF, p-value: < 2.2e-16

We see that the p-value associated with (coefficient of lstat) is very small, and thus reject . We conclude that lstat has a linear relationship with medv.

In general, a large p-value would indicate that, any linear association we see between and is most likely by chance even if and are not actually related. A small p-value would indicate that it is unlikely to observe a large association between and due to chance in absence of a real relationship.

Another way to test at is to check whether the value zero is in the confidence interval of or not.[[13]](#footnote-79) In the Boston data example with only lstat as predictor, zero is not in the confidence interval of , and thus the slope parameter is significantly different from zero with level .

Caution must be taken to interpret results from model with interaction terms. For example, let us investigate the summary of model fit with lstat, age and lstat:age interaction that we saw previously.

summary(int\_ols)

Call:  
lm(formula = medv ~ lstat \* age, data = Boston)  
  
Residuals:  
 Min 1Q Median 3Q Max   
-15.806 -4.045 -1.333 2.085 27.552   
  
Coefficients:  
 Estimate Std. Error t value Pr(>|t|)   
(Intercept) 36.0885359 1.4698355 24.553 < 2e-16 \*\*\*  
lstat -1.3921168 0.1674555 -8.313 8.78e-16 \*\*\*  
age -0.0007209 0.0198792 -0.036 0.9711   
lstat:age 0.0041560 0.0018518 2.244 0.0252 \*   
---  
Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
  
Residual standard error: 6.149 on 502 degrees of freedom  
Multiple R-squared: 0.5557, Adjusted R-squared: 0.5531   
F-statistic: 209.3 on 3 and 502 DF, p-value: < 2.2e-16

Note that the interaction term is statistically significant but the *main effect of age* is not significant. Thus we can not say age is not associated with even though the man effect is not significant. Also, we can not drop age from the model since age:lstat interaction needs to be in the model.

## test

In the multiple linear regression with predictors, we investigate the whether the linear model is at all needed by testing vs.  at least on is non-zero. We can use test to do so. In general, we can test for *any subset* of the predictors using -test, that is,

where we are testing the effects of the last predictors (last columns of ).

We reject if the observed value of is “large enough”. Equivalently, a formula of the p-value of this test is available as well.

Let us visit the interaction model with lstat, age and lstat:age as predictors – results were saved in the int\_ols object. Mathematically, we write the model as

where and correspond to lstat and age, respectively. Suppose we want to test the overall model, that is, jointly test the effects of all the three terms, . The F-test results can be found at the bottom of the summary output for this model:

summary(int\_ols)

Call:  
lm(formula = medv ~ lstat \* age, data = Boston)  
  
Residuals:  
 Min 1Q Median 3Q Max   
-15.806 -4.045 -1.333 2.085 27.552   
  
Coefficients:  
 Estimate Std. Error t value Pr(>|t|)   
(Intercept) 36.0885359 1.4698355 24.553 < 2e-16 \*\*\*  
lstat -1.3921168 0.1674555 -8.313 8.78e-16 \*\*\*  
age -0.0007209 0.0198792 -0.036 0.9711   
lstat:age 0.0041560 0.0018518 2.244 0.0252 \*   
---  
Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
  
Residual standard error: 6.149 on 502 degrees of freedom  
Multiple R-squared: 0.5557, Adjusted R-squared: 0.5531   
F-statistic: 209.3 on 3 and 502 DF, p-value: < 2.2e-16

Note the line: “F-statistic: 209.3 on 3 and 502 DF, p-value: < 2.2e-16”. The very small p-value indicates that we reject , and the model is useful in predicting .

Alternatively, we can fit two models: the full model (int\_ols) and another model with only intercept, and conduct -test ourselves using the anova() function in R.

# Full model already fitted: int\_ols  
# Reduced model with only intercept  
model\_red <- lm(medv ~ 1, data = Boston)  
anova(model\_red, int\_ols)

Analysis of Variance Table  
  
Model 1: medv ~ 1  
Model 2: medv ~ lstat \* age  
 Res.Df RSS Df Sum of Sq F Pr(>F)   
1 505 42716   
2 502 18978 3 23739 209.31 < 2.2e-16 \*\*\*  
---  
Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Notice that we have obtained identical -statistic and p-value compared to those in the summary output.

Now suppose we want to test “whether *age* have any association with response” or not. Since we have the interaction term, we have to test for both main and interaction effects of age. Thus we test . We take the second approach to do this.

# Full model already fitted: int\_ols  
# Reduced model with only intercept and lstat  
model\_without\_age <- lm(medv ~ lstat, data = Boston)  
anova(model\_without\_age, int\_ols)

Analysis of Variance Table  
  
Model 1: medv ~ lstat  
Model 2: medv ~ lstat \* age  
 Res.Df RSS Df Sum of Sq F Pr(>F)   
1 504 19472   
2 502 18978 2 494.67 6.5425 0.001567 \*\*  
---  
Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Here we have observed -statistic with a p-value of . If we use level , we reject , and can conclude that age does have association with response.

When , that is, we are testing for one predictor only, the -test and -test are quivalent. In fact, the square of the -statistic will give the -statistic. Check this from summary of simple\_ols when we test for only lstat.

So why do we need the -test when we can examine -test results for each predictor in our model? Can we not conclude the “model is useful in predicting response” if at least one -test gives significant result without performing overall -test? Indeed, we can not make such a claim. The main issue if *type I error* or *level* of the test. Typically, we set as level of the tests. That means, for individual -tests, there is a chance that some effect will be detected as significant when it is actually not significant. Thus, we might have about many false significant results,[[14]](#footnote-81) just by chance, even though none of the predictors might be useful. In fact, for large , it is very likely we will observe at least one false significance just by chance, and incorrectly conclude that the model is useful. However, the F-statistic does not suffer from this problem because it adjusts for the number of predictors. If is true, there is only a chance that the F-statistic detect a false significance regardless of the number of predictors.

# Evaluating model performance

Like any other learner, we need ways to evaluate model performance of linear regression. We will discuss how to assess model fit in the training data, and then in unseen test observations.

## Training set performance

We can measure how well the model fits the training data by using the following measures:

* Residual squared error (RSE).
* Coefficient of determination, .
* statistic discussed before

We have seen RSE as the estimator of in the previous sections. In general, RSE quantifies the uncertainty in prediction on from *even if the true regression parameters were known.* We can view RSE as the amount the response will deviate on average from the true regression line. A small RSE would indicate a good regression fit. In the Boston data example with only lstat as predictor described above, we have . Thus, even if we knew the true regression line (assuming that the linear model is correct), a prediction of medv based on lstat would still be off by units on average. In the Boston data, the mean value of medv over all values of lstat is 22.53. Thus we are making an error in the amount of 28 percent.

The RSE is considered a measure of the *lack of fit* of the model. Small values of RSE imply the predictions are close to the observed values which indicate good model fit. Large values of RSE would indicate that the model did not fit the data well. However, it is often not clear what values of RSE is acceptable. The coefficient of determination () is another option to measure goodness of fit.

TSS measures the total variance in the response[[15]](#footnote-84). We can think of TSS as the amount of variability inherent in the response before the regression is performed. In contrast, RSS measures the amount of variability that is left unexplained after performing the regression. Thus we can interpret as the *proportion of variance* in the response *explained by the model*. It can be shown that , with larger values indicting better fit. values close to zero would indicate that perhaps the linear model is wrong, and/or the error variance is high. Another way to interpret is that

## Test set performance

We can use the techniques and data splitting methods (CV, Bootstrap, holdout etc) to evaluate model performance on unseen test data. For example, the code below uses 5-fold CV, repeated 10 times, to estimate the test error for the model with lstat, age and lstat:age as predictors.

set.seed(1001)  
# control params  
cv <- trainControl(method = "repeatedcv",   
 number = 5,   
 repeats = 10)  
# training  
res <- train(medv ~ lstat \* age,   
 data = Boston,   
 method = "lm",   
 trControl = cv)  
res

Linear Regression   
  
506 samples  
 2 predictor  
  
No pre-processing  
Resampling: Cross-Validated (5 fold, repeated 10 times)   
Summary of sample sizes: 404, 405, 406, 404, 405, 405, ...   
Resampling results:  
  
 RMSE Rsquared MAE   
 6.269136 0.5467627 4.542126  
  
Tuning parameter 'intercept' was held constant at a value of TRUE

We can compare several regression models as well. In the code below, we fit five models, and compare their test RMSE values using 10 times repeated 5-fold CV. Note the use of the resample() function from caret package. Figure shows boxplots of estimated test errors using CV for the model.

set.seed(1001)  
# control params  
cv <- trainControl(method = "repeatedcv",   
 number = 5,   
 repeats = 10)  
# training  
model1 <- train(medv ~ lstat + age,   
 data = Boston,   
 method = "lm",   
 trControl = cv)  
  
model2 <- train(medv ~ lstat \* age,   
 data = Boston,   
 method = "lm",   
 trControl = cv)  
  
model3 <- train(medv ~ lstat + age + I(lstat^2) + I(age^2),   
 data = Boston,   
 method = "lm",   
 trControl = cv)  
  
model4 <- train(medv ~ lstat \* age + I(lstat^2) + I(age^2),   
 data = Boston,   
 method = "lm",   
 trControl = cv)  
  
model5 <- train(medv ~ .,   
 data = Boston,   
 method = "lm",   
 trControl = cv)  
  
# Comparison  
rsm <- resamples(list(model1, model2, model3, model4, model5))  
summary(rsm, metric = "RMSE")

Call:  
summary.resamples(object = rsm, metric = "RMSE")  
  
Models: Model1, Model2, Model3, Model4, Model5   
Number of resamples: 50   
  
RMSE   
 Min. 1st Qu. Median Mean 3rd Qu. Max. NA's  
Model1 5.339992 5.854950 6.136734 6.195250 6.542320 7.384349 0  
Model2 4.919853 5.910121 6.241548 6.272189 6.713179 7.531853 0  
Model3 4.379361 4.951992 5.399021 5.333451 5.710442 6.215090 0  
Model4 4.483109 4.989531 5.271986 5.234163 5.588847 6.136971 0  
Model5 3.889329 4.561428 4.865174 4.896241 5.279765 6.007630 0

# Comparison plots  
bwplot(rsm, metric = "RMSE")

|  |
| --- |
| Boxplots of estimated test RMSE for different models. |

Models 1 and 2 above are similar in test performance, as are model 3 and 4. But Models 3 and 4 are better that 1 and 2. Also, it seems that, as far as prediction accuracy is concerned, adding lstat:age interaction term does not improve prediction performance by much. Overall, model 5 (regression with main effects of all predictors) performs the best among the five model considered above.

# Model diagnostics

To estimate standard errors, and to perform inference, we needed certain assumptions on the errors and the model as a whole:

* The relationship between and ’s are indeed as posited by the linear regression model.
* Errors constant variance .
* Errors are normally distributed.

Other practical issue include:

* Multicollinearity among predictors
* Presence of influential points

For our inference to be valid, we need to make sure the assumptions mentioned above are satisfied. We present some diagnostics methods to address each of the issues mentioned above.

## Deviation from linearity

To evaluate whether the relationship posited by the fitted regression model actually captures the true relationship, we can use *residual plots*. For simple linear regression, we can plot the residuals vs. the predictor. For multiple linear regression, it is easier to plot residual vs. the predicted responses. If the model specification is adequate, there should be *no clear pattern* in the residual plot. In contrast, any pattern in the residual plot would indicate the model does not capture the relationship between and well. In the later case, transforming data (either or or both) might prove useful. Alternatively, a non-linear/non-parametric regression might be considered as well.

|  |
| --- |
| Residual vs Fitted values for two linear models fit on Boston data: regression of medv on lstat, age and their two-way interaction (left plot), and regression of log(medv) on lstat and lstat (right plot). |

In R, the plot() command will produce residual plot, along with other diagnostic plots for linear models.[[16]](#footnote-94) Figure shows residual plots for two models: regression of medv on lstat, age and their two-way interaction (left plot), and regression of log(medv) on lstat and lstat (right plot). the red lines are smooths of the plot to easily visualize any patterns in the scatterplots. We see the the left plot shows a non-linear pattern indicating that the fitted model is not adequate. The right plot shows little pattern suggesting a better fit.

## Non-constant Variance of Errors

|  |
| --- |
| Example of a residual plot with non-constant error variance |

The standard errors, confidence intervals, and hypothesis testing procedures discussed so far depends on the assumption of constant variance of the errors: . We call such errors *homoscedastic*. If errors have different variance, such phenomenon is called *heteroscedasticity*. In the residual plot (bottom panel) in Figure , the black dashed lines track the and quantiles of the residuals accross predicted values. We see that residual variability is slightly higher for smaller values of prediction, but overall the constant variance assumption seems reasonable here. If, for some other residual plot, we see a “megaphone shape” then constant variable assumption would be questionable, see for example, Figure .

## Normality of errors

Normality of errors are needed for development of confidence intervals and testing procedures discussed above. However, this assumption can be relaxed for large enough sample size. Usually, visual displays such as normal Q-Q plot of the residuals is used to check normality assumption. If the points align with the diagonal line well enough, we can conclude that the normality assumption is satisfactory. However, keep in mind that Q-Q plot is merely a visual tool, and often samples from non-normal distributions can produce normal like Q-Q plot (and vice-versa).

## Influential points

|  |
| --- |
| Normal Q-Q plot of residuals. |

Outliers and high leverage points can be detected using residual plots with *studentized residuals* and *leverage statistics*.

Recall our linear model is We estimate the regression parameter as Thus we we can predict the entire response vector by plugging-in as

where . The matrix is called the *hat matrix*. Detection of influential points depend on the following two results:

It can be shown that variance of the -th residual, , where is the -th diagonal entry of .

The -th diagonal entry of , , is called the *leverage* of the -th observation.

We define *studentized residuals* as residuals divided by their standard deviations. We can plot the studentized residuals against fitted values to detect outliers. Observations whose studentized residuals are quite far away from the rest[[17]](#footnote-104) are possible outliers. The function rstudent() can be used to compute studentized residuals. Figure shows an example of such a plot. The point with absolute residual of more than 4 might be a potential outlier.

|  |
| --- |
| Example of a plot of absolute studentized residuals vs. fitted values. |

|  |
| --- |
| Example of a plot of absolute studentized residuals vs. leverage. |

We can plot the studentized residuals vs. leverage statistics to detect possible high leverage points. Figure shows an example of such a leverage plot. It can be shown that value of leverage leverage statistic is always between and . Also, the average value of leverages for all the observations is , where is the number of columns in the model matrix . Thus, an observation can be a potential high leverage point if a given observation has a leverage statistic much larger that . In Figure , we have and , and thus . The point to the far right of the plot with leverage more that might be a high leverage point. We can obtain leverage statistics using the function hatvalues().

## Collinearity

Collinearity refers to high correlation between two or more predictors. Presence of such high correlation may lead to numerical instability of linear model fitting, reduce accuracy of estimation of regression coefficients, and reduce power of hypothesis tests.

Consider the two linear model fits for Boston data: (A) medv on tax and rad, and (B) medv on lstat and tax. The results are shown below.

| term | estimate | std.error | statistic | p.value |
| --- | --- | --- | --- | --- |
| **Model A** | | | | |
| (Intercept) | 35.6359 | 1.3465 | 26.4652 | 0.0000 |
| tax | -0.0386 | 0.0052 | -7.4847 | 0.0000 |
| rad | 0.2762 | 0.0997 | 2.7703 | 0.0058 |
| **Model B** | | | | |
| (Intercept) | 34.6128 | 0.5676 | 60.9848 | 0.0000 |
| lstat | -0.9326 | 0.0444 | -20.9986 | 0.0000 |
| rad | -0.0293 | 0.0364 | -0.8057 | 0.4208 |

Notice that in presence of tax the estimate and standard errors of rad changes drastically. This is because, tax and rad are highly correlated – Figure shows the correlation plot of Boston data, where we see that indeed tax and rad have high correlation.

|  |
| --- |
| Correlation plot of Boston data. |

If more that two predictors are closely related, we call the situation *multicollinearity*. Such situations can not be detected by simply inspecting the corrlation plot. Instead, we may look at the *variance inflation factor (VIF)*.

The minimum value of VIF is . As a rule of thumb, a VIF value larger than 5 or 10 indicates a problematic amount of multicollinearity. We can use car::vif() to calculate VIFs. In our example above, the VIF for models (A) and (B) are shown below.

Model A:

tax rad   
5.831426 5.831426

Model B:

lstat rad   
1.313723 1.313723

In presence of multicollinearity, we can exclude the problematic predictors. Alternatively, we can combine the collinear predictors., e.g., taking average.

# Prediction

As mentioned before, we can predict the response associated with a set of predictors as

In order to quantify uncertainty of the prediction, we can use a *prediction interval*. We can use the function predict() to compute both the prediction and the corresponding prediction interval. The following code produces both for the new data point for the simple linear regression of medv on lstat: fit is the point predcition, lwr and upr are the lower and upper bound of the prediction interval, respectively. Note the function argument interval = "prediction".[[18]](#footnote-117)

simple\_ols <- lm(medv ~ lstat, data = Boston)  
pred\_int <- predict(simple\_ols,   
 newdata = data.frame(lstat = 5),  
 interval = "prediction",  
 level = 0.95)  
pred\_int

fit lwr upr  
1 29.80359 17.56567 42.04151

Note that the point prediction is simply estimate of . However, the prediction interval is not the same as the confidence interval of . This is because predicting the actual response is more difficult that estimating the mean . For the ideal case where we know the exact values of , then is exactly determined. But even then, the response has some variability due to error . Thus, even if we fully know the regression line, we can not predict the response exactly. The prediction interval captures this additional uncertainty. For example, the confidence interval of for the example shown above is as follows. Note the confidence interval is narrower than the prediction interval.

conf\_int <- predict(simple\_ols,   
 newdata = data.frame(lstat = 5),  
 interval = "confidence",  
 level = 0.95)

We can interpret these intervals as: when lstat = 5, we have confidence that the *mean* value of medv will fall in (29.01, 30.6) – this is the confidence interval. But medv of a *randomly chosen new observation* with fall in (17.57, 42.04).

# Including qualitative predictors

So far we have only discussed models where ’s are continuous variables. We can accommodate categorical predictors as well. To do so, we need to create new *binary* predictors representing each of the categories of the original predictors.

As an example, consider the variable *chas* (Charles River dummy variable, 1 = tract bounds river; 0 = otherwise.). This is a binary variable already coded 0/1. Suppose we write a linear model

|  |
| --- |
| Regression line for chas = 0 (solid red) and 1 (dashed black) based on a linear model with main effects of chas and lstat. |

|  |
| --- |
| Regression line for chas = 0 (solid red) and 1 (dashed black) based on a linear model with main effects of chas and lstat, as well as their interaction. |

where denotes medv, denotes lstat and denotes chas. This model effectively represents two lines, one for each value of chas. The display below gives the two regression lines:

Thus, the linear model above with only main effect of chas proposes linear relationship between medv and lstat where the two lines are parallel (same slope but possibly different intercept), see Figure . The solid red line corresponds to chas = 0 and the dashed black line for chas = 1.

If we include an interaction term:

then the two regression lines are allowed to have different slope as well as different intercept, as shown below:

See Figure for the fitted lines.

The ideas presented above can be generalized to categorical variables with more that two levels. Suppose that is a variables with three levels, “L1”, “L2” and “L3”. Then we need to create *dummy variables*:[[19]](#footnote-125)

$$
Z\_{i1} = I(Z\_i = "L1"), \,\, \mbox{and} \,\, Z\_{i2} = I(Z\_i = "L2")
$$

We do not need a new dummy for level “L3” since would encode “L3”.

Finally, we should be cautious when there are many categorical variables in the data. Since we need to expand each of them into multiple indicator variables, the number of predictors can increase by a lot. Thus we need to be careful if we intend use data splitting methods like CV. Consider for example the Ames housing data.[[20]](#footnote-126) We consider the variable Sale\_Price as response, and the rest as covariates. Out of 82 covariates, 42 are categorical. However, after expanding each categorical variable in to dummies, we will in fact have a total of predictors, not simply . If the sample size were smaller, say , standard techniques like 5-fold CV (training set size will be ) or split (training set size ) may produce unreliable results due to number of predictors being close to training set size. In general, we should always check the size of the model matrix before choosing a proper data splitting method.

# Subset selection

In practice, many of many of the variables is a dataset may not be associated with the response of interest. Including such irrelevant predictors in the model may lead to unnecessary complexity in the resulting model and therefore more variability in the estimates. Often we would like to remove the unnecessary variables before building our final model. Such a procedure will also help in interpretation of the model as well. This process of selecting relevant variables corresponding to a response is called *variable selection* or *feature selection*. In this section, we discuss methods to select a *subset* of the available covariates that we believe to be related to the response. Then the final model will be built by using least squares using the selected subset.

## Metrics for model selection

Usage of RSE and from the training set in model selection is undesirable as they will always choose the largest model possible – minimum RSE and maximum will occur when number of predictors is largest.

We can use the data splitting methods to estimate test errors, but sometimes they can be computationally expensive. Consider the Boston data with . If we want to investigate performance of all possible subsets, we have to go though models. On top of that if we want to use -fold CV, repeated 50 times, we have to fit a total of models.

Alternatively, there are metrics available that adjusts training performance metrics such as and to balance both goodness of fit and model complexity/size, so that a separate training set is not needed for model comparison. These approaches can be used to select among a set of models with different numbers of variables. Four such metrics are:

* Adjusted ,
* Akaike information criterion (AIC),
* Bayesian information criterion (BIC), and
* statistic.

Adjusted re-scales total sum of squares and RSS, before taking their ratio, to account for the number of predictors in the model. In contrast, AIC, BIC and *adds a penalty term* involving number of predictors to the training RSS to account for model size.

Suppose we have a model with predictors. Recall that . Adjusted is defined as

where is the number of predictors in the model. Maximizing the adjusted is equivalent to minimizing . Unlike , which monotonically decreases as increases, will increase and decrease as changes. We choose the model with maximum adjusted .

AIC, BIC and all have the form for a model with predictors:

where is a penalty term involving sample size, number of predictors in the model and estimated error variance using the full model containing all predictors. The three metrics use the following form of :[[21]](#footnote-128)

We choose the model which gives minimum AIC/BIC values.

|  |
| --- |
| Example of model selection using AIC/, BIC and adjusted . |

It seems AIC and are equivalent from the formula above – this happens for linear regression model using least squares and normal errors. However, AIC and BIC both have general forms involving *log-likelihood* values, and can be computed for general regression problems.

We can see from the penalty terms that BIC tend to have a higher penalty than AIC/ as increases. Thus BIC tends to produce smaller models compared to AIC/. Figure shows an example of model selection using AIC/, BIC and adjusted .

## Best subset selection

In this approach, we need to fit a separate least square model to *each* of the possible combination of the predictors in the dataset, that is, we need to fit all possible models. We can either use CV/holdout or AIC/BIC to choose the best model. The following algorithm shows the best subset selection procedure.

1. Start with the model with only intercept, and no other predictor. Denote the model by .
2. For , fit all models with predictors, and pick the best model (smallest RSE, largest etc.). Denote the resulting model as .
3. Among the models , choose the best model using AIC, BIC, adjusted or CV.

Note that we can use cross-validation for the entire set of possible models if we have such computational resources (for larger , this procedure can have tremendous computational burden). The algorithm above reduces this computational burden using Step 2, where it identifies the best model for each subset size *on the training set*. Thus we reduce the problem from possible models to possible models. However, performing CV, if possible, has the distinct advantage over AIC/BIC that it directly estimates the test error for each models.

In R, we can use regsubsets() in the leaps package to perform best subset selection. We demonstrate this procedure using Boston data. Note the usage of the argument nvmax = 11. This ensures that we will search of subsets up to size 12 (Since Boston data has 12 predictors).

library(leaps)  
# Best model for each model size  
bestmod <- regsubsets(medv ~ .,   
 data = Boston,  
 nvmax = 12)  
# summary  
mod\_summary <- summary(bestmod)  
mod\_summary

Subset selection object  
Call: regsubsets.formula(medv ~ ., data = Boston, nvmax = 12)  
12 Variables (and intercept)  
 Forced in Forced out  
crim FALSE FALSE  
zn FALSE FALSE  
indus FALSE FALSE  
chas FALSE FALSE  
nox FALSE FALSE  
rm FALSE FALSE  
age FALSE FALSE  
dis FALSE FALSE  
rad FALSE FALSE  
tax FALSE FALSE  
ptratio FALSE FALSE  
lstat FALSE FALSE  
1 subsets of each size up to 12  
Selection Algorithm: exhaustive  
 crim zn indus chas nox rm age dis rad tax ptratio lstat  
1 ( 1 ) " " " " " " " " " " " " " " " " " " " " " " "\*"   
2 ( 1 ) " " " " " " " " " " "\*" " " " " " " " " " " "\*"   
3 ( 1 ) " " " " " " " " " " "\*" " " " " " " " " "\*" "\*"   
4 ( 1 ) " " " " " " " " " " "\*" " " "\*" " " " " "\*" "\*"   
5 ( 1 ) " " " " " " " " "\*" "\*" " " "\*" " " " " "\*" "\*"   
6 ( 1 ) " " " " " " "\*" "\*" "\*" " " "\*" " " " " "\*" "\*"   
7 ( 1 ) " " "\*" " " "\*" "\*" "\*" " " "\*" " " " " "\*" "\*"   
8 ( 1 ) "\*" "\*" " " "\*" "\*" "\*" " " "\*" " " " " "\*" "\*"   
9 ( 1 ) "\*" "\*" " " " " "\*" "\*" " " "\*" "\*" "\*" "\*" "\*"   
10 ( 1 ) "\*" "\*" " " "\*" "\*" "\*" " " "\*" "\*" "\*" "\*" "\*"   
11 ( 1 ) "\*" "\*" " " "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*"   
12 ( 1 ) "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*"

The summary shows, for each model size, which predictors give the best model (based on training set performance). Now we can use either AIC/BIC or adjusted to choose the best model among these 12 models.

metrics <- data.frame(aic = mod\_summary$cp,  
 bic = mod\_summary$bic,  
 adjR2 = mod\_summary$adjr2)  
metrics

aic bic adjR2  
1 343.848074 -385.0521 0.5432418  
2 170.658081 -496.2582 0.6371245  
3 98.320999 -549.4767 0.6767036  
4 78.641892 -561.9884 0.6878351  
5 47.647706 -585.6823 0.7051702  
6 35.388139 -592.9553 0.7123567  
7 30.246610 -593.6275 0.7156820  
8 24.822922 -594.6734 0.7191751  
9 18.162742 -597.0648 0.7233609  
10 9.120223 -602.0442 0.7288734  
11 11.046965 -595.8928 0.7283649  
12 13.000000 -589.7145 0.7278399

The minimum AIC/BIC as well as maximum adjusted occurs for model size . The best fitted model is below.

round( coef(bestmod, 10), 2)

(Intercept) crim zn chas nox rm   
 41.45 -0.12 0.05 2.87 -18.26 3.67   
 dis rad tax ptratio lstat   
 -1.52 0.28 -0.01 -0.93 -0.55

As mentioned before, investigating all off the models can be computationally intensive for large values of . The following two approaches provide computationally efficient alternatives using *stepwise subset selection*.

## Forward Stepwise Selection

Recall that the best subset selection procedure considers all possible models containing subsets of the predictors. In contrast, *forward stepwise selection* considers a much smaller set of models. The algorithm as as follows:

|  |
| --- |
| AIC, BIC and Adjusted for best subset selection in Boston data. |

1. Start with the model with only intercept, and no other predictor. Denote the model by .
2. For ,
   * consider all models that adds one more predictor to the existing predictors in .
   * choose the best among these models; denote this model by
3. Among the models , choose the best model using AIC, BIC, adjusted or CV.

Forward stepwise selection involves fitting one intercept-only model, along with models in the th iteration, for . This reduces the computational complexity substantially from the best subset selection, which fits models for each . We should keep in mind that, since forward stepwise selection does not go through all possible models, there is no assurance that it will find the best model.

The following code performs forward stepwise selection for Boston data example.

forward <- regsubsets(medv ~ .,   
 data = Boston,  
 nvmax = 12,  
 method = "forward")  
# summary  
mod\_summary <- summary(forward)  
mod\_summary

Subset selection object  
Call: regsubsets.formula(medv ~ ., data = Boston, nvmax = 12, method = "forward")  
12 Variables (and intercept)  
 Forced in Forced out  
crim FALSE FALSE  
zn FALSE FALSE  
indus FALSE FALSE  
chas FALSE FALSE  
nox FALSE FALSE  
rm FALSE FALSE  
age FALSE FALSE  
dis FALSE FALSE  
rad FALSE FALSE  
tax FALSE FALSE  
ptratio FALSE FALSE  
lstat FALSE FALSE  
1 subsets of each size up to 12  
Selection Algorithm: forward  
 crim zn indus chas nox rm age dis rad tax ptratio lstat  
1 ( 1 ) " " " " " " " " " " " " " " " " " " " " " " "\*"   
2 ( 1 ) " " " " " " " " " " "\*" " " " " " " " " " " "\*"   
3 ( 1 ) " " " " " " " " " " "\*" " " " " " " " " "\*" "\*"   
4 ( 1 ) " " " " " " " " " " "\*" " " "\*" " " " " "\*" "\*"   
5 ( 1 ) " " " " " " " " "\*" "\*" " " "\*" " " " " "\*" "\*"   
6 ( 1 ) " " " " " " "\*" "\*" "\*" " " "\*" " " " " "\*" "\*"   
7 ( 1 ) " " "\*" " " "\*" "\*" "\*" " " "\*" " " " " "\*" "\*"   
8 ( 1 ) "\*" "\*" " " "\*" "\*" "\*" " " "\*" " " " " "\*" "\*"   
9 ( 1 ) "\*" "\*" " " "\*" "\*" "\*" " " "\*" "\*" " " "\*" "\*"   
10 ( 1 ) "\*" "\*" " " "\*" "\*" "\*" " " "\*" "\*" "\*" "\*" "\*"   
11 ( 1 ) "\*" "\*" " " "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*"   
12 ( 1 ) "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*"

As before, the summary shows which predictors give the best model (based on training set performance) for each model size. Next we can choose the best model among these 12 models.

metrics <- data.frame(aic = mod\_summary$cp,  
 bic = mod\_summary$bic,  
 adjR2 = mod\_summary$adjr2)  
metrics

aic bic adjR2  
1 343.848074 -385.0521 0.5432418  
2 170.658081 -496.2582 0.6371245  
3 98.320999 -549.4767 0.6767036  
4 78.641892 -561.9884 0.6878351  
5 47.647706 -585.6823 0.7051702  
6 35.388139 -592.9553 0.7123567  
7 30.246610 -593.6275 0.7156820  
8 24.822922 -594.6734 0.7191751  
9 20.089732 -595.1344 0.7223035  
10 9.120223 -602.0442 0.7288734  
11 11.046965 -595.8928 0.7283649  
12 13.000000 -589.7145 0.7278399

round( coef(forward, 10), 2)

(Intercept) crim zn chas nox rm   
 41.45 -0.12 0.05 2.87 -18.26 3.67   
 dis rad tax ptratio lstat   
 -1.52 0.28 -0.01 -0.93 -0.55

|  |
| --- |
| AIC, BIC and Adjusted for forward stepwise selection in Boston data. |

## Backward Stepwise Selection

Like forward selection, backward selection also considers a smaller set of models. It start from including all the predictors, and gradually removes one predictor at a time. The following algorithm performs backward stepwise selection.

1. Start with the model with all the predictors included. Denote the model by .
2. For ,
   * consider all models that contain all but one of the predictors in , for a total of predictors.
   * choose the best among these models; denote this model by
3. Among the models , choose the best model using AIC, BIC, adjusted or CV.

Like forward stepwise selection, backward stepwise selection is not guaranteed to yield the best model containing a subset of the predictors. The following code performs backward stepwise selection for Boston data example.

backward <- regsubsets(medv ~ .,   
 data = Boston,  
 nvmax = 12,  
 method = "backward")  
# summary  
mod\_summary <- summary(backward)  
mod\_summary

Subset selection object  
Call: regsubsets.formula(medv ~ ., data = Boston, nvmax = 12, method = "backward")  
12 Variables (and intercept)  
 Forced in Forced out  
crim FALSE FALSE  
zn FALSE FALSE  
indus FALSE FALSE  
chas FALSE FALSE  
nox FALSE FALSE  
rm FALSE FALSE  
age FALSE FALSE  
dis FALSE FALSE  
rad FALSE FALSE  
tax FALSE FALSE  
ptratio FALSE FALSE  
lstat FALSE FALSE  
1 subsets of each size up to 12  
Selection Algorithm: backward  
 crim zn indus chas nox rm age dis rad tax ptratio lstat  
1 ( 1 ) " " " " " " " " " " " " " " " " " " " " " " "\*"   
2 ( 1 ) " " " " " " " " " " "\*" " " " " " " " " " " "\*"   
3 ( 1 ) " " " " " " " " " " "\*" " " " " " " " " "\*" "\*"   
4 ( 1 ) " " " " " " " " " " "\*" " " "\*" " " " " "\*" "\*"   
5 ( 1 ) " " " " " " " " "\*" "\*" " " "\*" " " " " "\*" "\*"   
6 ( 1 ) "\*" " " " " " " "\*" "\*" " " "\*" " " " " "\*" "\*"   
7 ( 1 ) "\*" " " " " " " "\*" "\*" " " "\*" "\*" " " "\*" "\*"   
8 ( 1 ) "\*" " " " " " " "\*" "\*" " " "\*" "\*" "\*" "\*" "\*"   
9 ( 1 ) "\*" "\*" " " " " "\*" "\*" " " "\*" "\*" "\*" "\*" "\*"   
10 ( 1 ) "\*" "\*" " " "\*" "\*" "\*" " " "\*" "\*" "\*" "\*" "\*"   
11 ( 1 ) "\*" "\*" " " "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*"   
12 ( 1 ) "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*"

As before, the summary shows which predictors give the best model (based on training set performance) for each model size. Next we can choose the best model among these 12 models.

metrics <- data.frame(aic = mod\_summary$cp,  
 bic = mod\_summary$bic,  
 adjR2 = mod\_summary$adjr2)  
metrics

aic bic adjR2  
1 343.848074 -385.0521 0.5432418  
2 170.658081 -496.2582 0.6371245  
3 98.320999 -549.4767 0.6767036  
4 78.641892 -561.9884 0.6878351  
5 47.647706 -585.6823 0.7051702  
6 43.504080 -585.2278 0.7079301  
7 36.420912 -587.6576 0.7123077  
8 27.487790 -592.0508 0.7177158  
9 18.162742 -597.0648 0.7233609  
10 9.120223 -602.0442 0.7288734  
11 11.046965 -595.8928 0.7283649  
12 13.000000 -589.7145 0.7278399

coef(forward, 10)

(Intercept) crim zn chas nox rm   
 41.45174748 -0.12166488 0.04619119 2.87187265 -18.26242664 3.67295747   
 dis rad tax ptratio lstat   
 -1.51595105 0.28393226 -0.01229150 -0.93096144 -0.54650916

|  |
| --- |
| AIC, BIC and Adjusted for backward stepwise selection in Boston data. |

In the Boston data example seen so far, results from using AIC, BIC and adjusted match – they all choose the same model. This is not the case in general setting. We might have different “best” models depending on the evaluation criterion we use. In that case, we will just pick the criteria we like the most (e.g. BIC for typically giving smaller models), and go with the corresponding best model.

## Using the holdout and Cross-Validation for subset selection

As mentioned before, apart from AIC/BIC/adjusted , it is also possible to use data splitting techniques such as holdout or CV for model selection. Ideally, we can run CV for each of the models, and choose the one with best test error. However, such an approach can be computationally expensive.

Alternatively, we can use the algorithms presented above and use CV on them. It is important to recall our discussion in the previous chapters about proper implementation of CV: the entire model building process, including any tuning, has to be applied to the training set. We **can not** simply use steps 1 and 2 on the full data to get and then just use CV on the final models. The following paragraph is quoted verbatim from the textbook to emphasize this important point.

In order for these approaches to yield accurate estimates of the test error, we must use *only the training observations* to perform all aspects of model-fitting—including variable selection. Therefore, the determination of which model of a given size is best must be made using *only the training observations*. This point is subtle but important. If the full data set is used to perform the best subset selection step, the validation set errors and cross-validation errors that we obtain will not be accurate estimates of the test error.

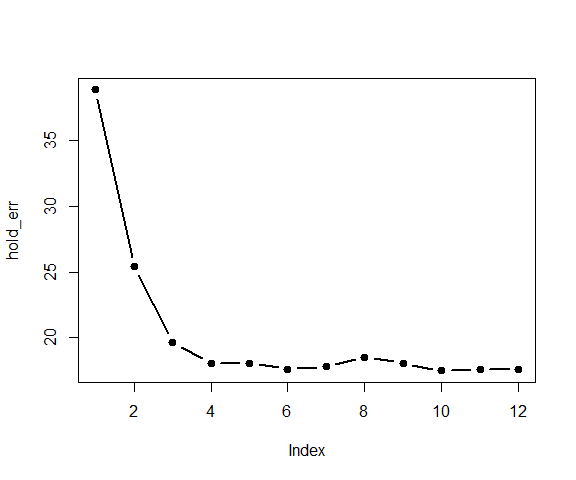
— Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani, An Introduction to Statistical Learning, second edition, 2021, page 271.

Thus we can think the *model size* as tuning parameter here, since each training set might yield different models even if the size remains the same. We use holdout/CV to choose the best model size, and then choose the best model of that size using the full data.

The algorithm of subset selection using *holdout method* is as follows:

* Split the observations into training and test sets.
* Apply best/forward/backward selection method on the training set.
* For *each model size*, pick the best model, and compute test error using test set.
* Choose the optimal model size that has minimum test error.
* Finally, perform best/forward/backward subset selection on the full data set, and select the best model of the size chosen in the previous step.

set.seed(1001)  
## Create test and training sets  
library(rsample)  
data\_split <- initial\_split(Boston, prop = 0.8)  
test\_set <- testing(data\_split)  
train\_set <- training(data\_split)  
  
## Best subset selection on the training data  
best\_train <- regsubsets(medv ~ .,   
 data = train\_set,  
 nvmax = 12)  
train\_sum <- summary(best\_train)  
  
## For each model size, estimate test performance  
# A function to predict and estimate error   
# on the test set. Inputs are   
# model size (mod\_size),  
# summary outout of the selection process (reg\_summary)  
# model matrix of the test data (test\_model)  
# test set response (test\_resp)  
test\_err <- function(mod\_size,   
 reg\_summary,   
 test\_model,  
 test\_resp){  
 # get regression coefs  
 betahat <- coef(reg\_summary$obj, mod\_size)  
 # get best subset of the specified size  
 sub <- reg\_summary$which[mod\_size, ]  
 # Create test model matrix, predcition, test error  
 model <- test\_model[, sub]  
 yhat <- model %\*% betahat  
 err <- mean((test\_resp - yhat)^2)  
 return(err)  
}  
  
## Apply the function above to each model size  
test\_model <- model.matrix(~ . - medv, data = test\_set)  
test\_resp <- test\_set$medv  
hold\_err <- sapply(1:12, test\_err,   
 reg\_summary = train\_sum,  
 test\_model = test\_model,   
 test\_resp = test\_resp)  
plot(hold\_err, type = 'b', pch=19, lwd=2)



## Best model size and refit of full data  
size\_opt <- which.min(hold\_err)  
bestmod <- regsubsets(medv ~ .,  
 data = Boston,  
 nvmax = 12)  
coef(bestmod, size\_opt)

(Intercept) crim zn chas nox rm   
 41.45174748 -0.12166488 0.04619119 2.87187265 -18.26242664 3.67295747   
 dis rad tax ptratio lstat   
 -1.51595105 0.28393226 -0.01229150 -0.93096144 -0.54650916

In this particular example, we have the same 10-variable model as before. We refit the full data set in order to obtain more accurate estimates of the regression coefficient estimates. It is important that we perform best/forward/backward subset selection on the full data set and select the best model with 10 variables (for this example), rather than simply using the variables that were obtained from the training set. This is because the best model with 10 predictors on the full data set may be different from the corresponding model on the training set.

We can similarly use -fold cross-validation as follows:

* Split the data into equally sized folds.
* For :
  + Set -th fold as test set, and the remaining folds as training set.
  + Apply best/forward/backward selection method on the training set.
  + For *each model size*, pick the best model, and compute test error using test set.
* Choose the optimal model size that has minimum average test error over folds.
* Finally, perform best/forward/backward subset selection on the full data set, and select the best model of the size chosen in the previous step.

|  |
| --- |
| Best subset selection using 5-fold cross-validation. The gray lines are test MSE profiles for the 5 folds. The black line is the mean test MSE over the folds. The red error bars indicate test MSE +/- one SE. |

Figure shows the results best subset selection using a 10-fold CV. The resulting model has size 10, and in fact is the same as the one chosen by holdout in this example.

Notice that even though the model with 10 predictors give the lowest test MSE, the models containing 5 – 9 predictors also have similar (slightly higher) MSE values. Surely, if we repeated CV using different folds, the exact minimum might change. In this setting, we often use the *one-standard-error rule*: calculate the standard error of the estimated test MSE from the 10 folds for each model size, and then select the smallest model for which the estimated test error is within one standard error of the minimum estimated MSE. If a set of models are essentially equal in performance, then one-standard-error rule would chose the the model with the smallest number of predictors. In our example in Figure , one-standard-error rule chooses a model with 5 predictors:

(Intercept) nox rm dis ptratio lstat   
 37.4991961 -17.9965715 4.1633074 -1.1846623 -1.0457738 -0.5810836

As a final note on correctly implementing cross-validation in general, we quote the following paragraph verbatim from *Elements of Statistical Learning*, **Section 7.10.2: The Wrong and Right Way to Do Cross-validation**:

Consider a classification problem with a large number of predictors, as may arise, for example, in genomic or proteomic applications. A typical strategy for analysis might be as follows:

1. Screen the predictors: find a subset of “good” predictors that show fairly strong (univariate) correlation with the class labels
2. Using just this subset of predictors, build a multivariate classifier.
3. Use cross-validation to estimate the unknown tuning parameters and to estimate the prediction error of the final model.

Is this a correct application of cross-validation? Consider a scenario with N = 50 samples in two equal-sized classes, and p = 5000 quantitative predictors (standard Gaussian) that are independent of the class labels. The true (test) error rate of any classifier is 50%. We carried out the above recipe, choosing in step (1) the 100 predictors having highest correlation with the class labels, and then using a 1-nearest neighbor classifier, based on just these 100 predictors, in step (2). Over 50 simulations from this setting, the average CV error rate was 3%. This is far lower than the true error rate of 50%.

What has happened? The problem is that the predictors have an unfair advantage, as they were chosen in step (1) on the basis of all of the samples. Leaving samples out after the variables have been selected does not correctly mimic the application of the classifier to a completely independent test set, since these predictors “have already seen” the left out samples.

Even though the discussion above is in the context of classification, the idea still applies to regression problems. Instead of misclassification error rate, we will be concerned about test MSE.

If we do need to screen predictors for a specific regression model, we need to do so *without involving response*, that is, using *unsupervised* methods. This should be done *before splitting data*. Again we quote a paragraph from *Elements of Statistical Learning*:

In general, with a multistep modeling procedure, cross-validation must be applied to the entire sequence of modeling steps. In particular, samples must be “left out” before any selection or filtering steps are applied. There is one qualification: initial unsupervised screening steps can be done before samples are left out. For example, we could select the 1000 predictors with highest variance across all 50 samples, before starting cross-validation. Since this filtering does not involve the class labels, it does not give the predictors an unfair advantage.

# Regularization/Shrinkage methods

Another approach to selecting relevant predictors is to fit a model with all predictors but put *constraints* on the regression coefficients. This is called *regularization* of the estimates. It is done is such a way that the resulting estimates are pulled towards zero – this is called *shrinkage*. Without going into mathematical details, it can be shown that shrinking the coefficients towards zero in this manner increases their bias but significantly reduces their variance.

A common regularization method is to add an extra *penalty term* to the usual least squares criterion. In other words, we minimize a criterion of the form

where the term is a penalty term involving the regression coefficients. Depending on the form of the penalty terms, we have different regression methods. In this section, we will discuss several such estimation methods.

## Ridge regression

Ridge regression shrinks the regression coefficients towards zero by imposing a *quadratic penalty*. The ridge regression coefficient estimates are obtained by minimizing[[22]](#footnote-153)

where is a tuning parameter. The penalty term is called a *shrinkage penalty*.[[23]](#footnote-154) Here controls the relative impact of the two terms on the regression coefficient estimates. For large values of , the quadratic penalty term dominates the criterion, and the resulting estimates approach to zero. When , there is no penalty, and thus we get exactly the ordinary least squares estimates. Thus we must select a reasonable value of to balance both the terms.  
Recall that denotes the model matrix of the regression problem. We can show that ridge regression solutions have a closed form expression (if we also penalize intercept):[[24]](#footnote-155)

$$
\widehat\beta^{\rm ridge} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I})^{-1}\mathbf{X}^T\mathbf{Y}.
$$

Notice again that setting gives us the least squares estimates, . Also note that, for , the matrix always has an inverse *even if does not have full column rank*. Thus, even in presence of collinearity/redundant columns in , ridge regression will still produce unique regression estimates.[[25]](#footnote-156)

Figure shows the estimated ridge regression coefficients for different values of in Boston data with medv as response, and *standardized* predictors. The left most part of the plot corresponds to , and shows the least squares estimates. The right extreme of the plot represents a large value of , and we see that all the coefficients are very close to zero.

|  |
| --- |
| Ridge regression coefficients for different values of lamda (log10 scale) for Boston data. |

We can also view the ridge regression problem as a *constrained minimization problem*,

subject to the constraint

for some . The second formulation of ridge regression explicitely puts constraint on the size of the regression coefficients. The parameters in the penalized formulation and in the constraint formulation are connected via an one-to-one relationship.

Based on the second formulation, we can think of ridge regression as minimizing RSS of a linear regression while preventing the regression coefficients from getting too large or small. The parameter determines how large/small regression coefficients can become. If is set to very large, then we are effectively allowing ’s to take any value (equivalent to setting a small ). On the other hand, a small will force the ’s to be smaller and closer to zero (equivalent to setting large ).

In presence of multicollinearity, the corresponding ’s can become wildly variable. A very large positive on one variable can be canceled by a similarly large negative on another predictor correlated to the first one. A size constraint imposed by , fixes this issue.

Before fitting the ridge regression model, we need to aware that scaling the predictors is often needed. In least squares estimation, scaling/standardizing a predictor does not *not* change the overall quality of the fit (e.g., , etc). If we multiply a predictor by a constant , then the resulting least square coefficient estimate will get multiplied by . In other words, using least squares, the quantity will remain the same no matter how we scale the -th predictor.[[26]](#footnote-160)

mod1 <- lm(medv ~ lstat, data = Boston)  
mod2 <- lm(medv ~ I(5\*lstat), data = Boston)  
# Coefficients  
cbind(original = mod1$coefficients[2],  
 scaled = mod2$coefficients[2])

original scaled  
lstat -0.9500494 -0.1900099

In contrast, ridge regression estimates can change substantially depending on scaling of the predictors. In fact, ridge regression estimators $\widehat\beta^{\rm ridge}\_j$ will depend on the scaling of the -th predictor, the value of the tuning parameter , *and* the scaling of the *other* predictors as well. Therefore it is best to apply ridge regression *after we have standardized each of the predictors*. This way, each predictor has variance 1, and the final fit will not depend on the scale on which the predictors are measured.

In addition, the ridge formulation does not penalize the intercept . This is due to the fact that the ridge estimates depend on the center chosen for the responses. Specifically, in least squares regression, if we add a constant to each of the responses , the resulting predictions also shift by the same amount . But this does not happen in ridge regression if we penalize the intercept – therefore we do not penalize .

It can be shown that, if we center each covariate, that is, we use as predictors, then the estimator of the intercept is simply the sample mean of : . The remaining coefficients, , are estimated by a ridge regression without intercept.

For simplicity, we will henceforth assume that the model matrix $\X$ does not include intercept, and thus has only columns, not . We will also assume that mean of each column is zero.

Under this assumption, we still have the same form of the solution: $(\widehat\beta\_1, \ldots, \widehat\beta\_p) = (\X^T\X + \lambda \mathbf{I})^{-1} \X^T\Y$. Furthermore, if we standardize predictors beforehand and if they are orthogonal to each other, it can be shown that $\widehat\beta\_j^{\rm ridge} = \widehat\beta/(1 + \lambda)$.

In R, we can use the glmnet() function in the glmnet library.[[27]](#footnote-161) Let us use the Boston data for example. Note the usage of alpha = 0 (ensures we are fitting ridge regression as glmnet() can fit other models like LASSO and elastic net as well).

library(glmnet)  
## model matrix (standardized) and response  
medv <- Boston$medv  
model\_mat <- Boston[ , -13]  
model\_mat <- scale(model\_mat)  
model\_mat <- as.matrix(model\_mat)

## Fit ridge regression for a grid of lambda  
grid <- 10^seq(-2, 10, length = 100)  
boston\_ridge <- glmnet(y = medv, x = model\_mat,  
 alpha = 0,  
 lambda = grid)  
betahat <- coef(boston\_ridge)

dim(betahat)

[1] 13 100

We constructed the model matrix by excluding intercept since it will be automatically included by glmnet() as well as excluding medv. Here we have used a custom grid of values.[[28]](#footnote-162) For each value of , the output betahat contains the corresponding estimates of the regression coefficients. Figure shows the estimated coefficients for different values of .

How do we choose the “optimal” value of ? We again come back to *bias-variance trade-off*. Note that the penalty parameter effectively controls the model complexity: small values of results in close to least squares fit (lower bias, higher variance), while large values of results in almost an intercept-only model (higher bias, lower variance). Figure shows bias-variance trade-off of ridge regression.

|  |
| --- |
| Bias-variance trade-off of ridge regression. Figure taken from extit{Introduction to Statistical Learning}. Displayed are squared bias (black), variance (green), and test mean squared error (purple) for the ridge regression predictions on a simulated data set. The horizontal dashed lines indicate the minimum possible MSE. |

Ideally, we would like to select that minimizes test MSE. We can use data splitting methods such as cross-validation (or holdout) to do so. We choose a grid of candidate values of , and compute the cross-validation (or holdout) error for each value. The optimal is the one with minimum test error. Finally, we refit the model to the full data using the optimal .

We can use glmnet.cv() function to perform cross-validation. By default, glmnet.cv() uses 10-fold CV.[[29]](#footnote-166)

set.seed(1001)  
grid <- 10^seq(-2, 10, length = 100)  
cv\_out <- cv.glmnet(x = model\_mat, y = medv,   
 alpha = 0,   
 lambda = grid)

We can plot the results from CV process using the output of cv.glmnet() output. Figure shows the results.

# Plot cv results  
plot(cv\_out)

|  |
| --- |
| Cross-validation results for Boston data using ridge regression. |

The “best” value of can chosen by minimizing the CV error. The left vertical line in Figure represents this value. From Figure , we see that there are a range of values that give similar CV errors, and the dip in CV errors is not very pronounced. This suggests that we might just as well use least squares estimate in this case. Alternatively, we can also us the *one standard error* rule to choose : rather than choosing the that gives the minimum test MSE, we would pick the largest (less model complexity) whose test MSE is within one standard error of the minimum test MSE. The right vertical line in Figure represents this value. The two values are shown below, along with the estimated coefficients as well as estimated least squares coefficients for comparison.

## lambda with minimum CV error/1 - SE  
bestlam <- data.frame(min = cv\_out$lambda.min,  
 one\_se = cv\_out$lambda.1se)  
bestlam

min one\_se  
1 0.04037017 2.656088

## Refit ridge regression  
# The cv\_out object already has the full data fit  
# for each lambda  
ridge\_min = predict(cv\_out$glmnet.fit,   
 type = "coefficients",   
 s = bestlam$min)  
ridge\_1se = predict(cv\_out$glmnet.fit,   
 type = "coefficients",   
 s = bestlam$one\_se)  
# Least squares  
ols <- coef(lm(medv ~ model\_mat))  
betahat <- cbind(ridge\_min, ridge\_1se, ols)  
colnames(betahat) <- c("min", "1se", "ols")  
betahat

13 x 3 sparse Matrix of class "dgCMatrix"  
 min 1se ols  
(Intercept) 22.53280632 22.5328063 22.53280632  
crim -1.02358656 -0.7085400 -1.04412968  
zn 1.06108439 0.5188476 1.09530317  
indus 0.03927427 -0.4800640 0.09239314  
chas 0.72873407 0.7558299 0.72134140  
nox -2.10742277 -0.8332851 -2.17363599  
rm 2.59148348 2.6550878 2.57025715  
age 0.08566619 -0.1792700 0.10163739  
dis -3.07829358 -1.3983445 -3.13909506  
rad 2.35789785 0.3916933 2.51992023  
tax -1.98858081 -0.6041403 -2.13738455  
ptratio -2.01044117 -1.5536893 -2.02970765  
lstat -3.91107337 -2.8416725 -3.94200236

## norm of betahat  
sqrt( colSums(betahat^2) )

min 1se ols   
23.66276 23.02258 23.71326

|  |
| --- |
| Predictors arranged by absolute values of their estimated coefficients using 1-SE rule. |

In general, when the true relationship between predictors and response is linear, least squares estimates will have low bias but can have high variance, especially when is close to . When , least squares estimates are not unique. In contrast, ridge regression will still perform well by trading off a small increase in bias for a large decrease in variance. Thus, ridge regression works best in situations where the least squares estimates have high variance.

A major disadvantage of ridge regression is that it does not exclude any variables from the final fitted model, that is, it always produces non-zero estimates of the regression coefficients. Ridge regression will not set any coefficients to exactly zero for any finite value of . Thus ridge regression can not be considered as a *variable selection* method. This is not a problem for prediction, but interpreting of a model fit with many small but non-zero coefficients can be difficult.

## Lasso regression

The lasso regression is another shrinkage method like ridge regression, but LASSO uses a penalty term involving sum of the absolute values of the regression coefficients, instead of sum of their squares. In particular, LASSO estimates of are obtained by minimizing

for . Due to the penalty term, there is no closed form solution to the lasso problem.[[30]](#footnote-174) An equivalent way to write the LASSO problem is in the form of a constrained minimization problem,

subject to the constraint

for some .

Much like ridge regression, lasso also shrinks the regression coefficients towards zero. However, due to the penalty term , some of the coefficients will be shrunk exactly to zero. It is easier to see if we have standardized the predictors, and if they are orthogonal to each other. In that case, the explicit lasso solution is $\widehat\beta\_j^{\rm lasso} = sign(\widehat\beta\_j)(|\widehat\beta\_j| - \lambda)\_+$. Thus lasso does perform variable selection. As a result, models generated from the lasso are generally much easier to interpret than those produced by ridge regression. In other words, lasso generates *sparse models* – some coefficients are estimated to be *exactly zero*.

From the point of view of the constrained formulation, for large values of , we will effectively get the least squares estimates. Specifically, it can be shown that if is chosen larger that , then lasso estimates are identical to least squares estimates. On the other hand, if we chose , then the least squares estimates are shrunk, on average, by about . Figure shows the reason some lasso estimates are exactly set to zero while ridge estimates are not. Here represents least squares solution while while the blue diamond and circle represent the lasso and ridge regression constraints. For large values of , the constraint region will contain and thus both ridge and lasso estimates will be identical to least squares (equivalently choosing ). For smaller values of , the least squares estimate may lie outside the constraint region, like we see in Figure .

|  |
| --- |
| Contours of the error and constraint functions for the lasso (left) and ridge regression (right). The solid blue areas are the constraint regions for lasso and ridge, while the red ellipses are the contours of the RSS. Figure taken from . |

The ridge and lasso estimates are the points where the contours (ellipses) of the RSS intersect with the corresponding constraint region. Since the constraint region of ridge regression is circular with no sharp points, this intersection will not generally occur on an axis. Thus ridge regression coefficient estimates will be non-zero. On the other hand, the lasso constraint region has corners at each of the axes. So the ellipse will often intersect the constraint region at an axis. When this occurs, one of the coefficients will equal zero. In higher dimensions, many of the coefficient estimates may equal zero simultaneously. In Figure , we have .

In R, we can use glmnet() with argument alpha=1 to fit lasso regression. The code presented in the ridge regression section will work here with only change being alpha=1. The lasso fit for Boston data is done below. Figure shows the estimated regression coefficients as changes. The left extreme of the plot corresponds to least squares fit ().

|  |
| --- |
| Lasso regression coefficients for different values of lambda (log10 scale) for Boston data. |

library(glmnet)  
## model matrix (standardized) and response  
medv <- Boston$medv  
model\_mat <- Boston[ , -13]  
model\_mat <- scale(model\_mat)  
model\_mat <- as.matrix(model\_mat)  
  
## Fit lasso regression for a grid of lambda  
grid <- 10^seq(-3, 7, length = 100)  
boston\_lasso <- glmnet(x = model\_mat, y = medv,  
 alpha = 1,  
 lambda = grid)  
beta\_hat <- coef(boston\_lasso)  
dim(beta\_hat)

[1] 13 100

Like ridge regression, we need to carefully select . We can use cross-validation (or holdout) methods to do so, as before.

## Lasso cross-validation  
set.seed(1001)  
grid <- 10^seq(-3, 7, length = 100)  
cv\_out <- cv.glmnet(x = model\_mat, y = medv,   
 alpha = 1,  
 lambda = grid)

# Plot cv results  
plot(cv\_out)

|  |
| --- |
| Cross-validation results for Boston data using lasso regression. |

Figure shows the results of selection of using 10-fold cross-validation. The values with minimum CV error and chosen by the one standard rule are shown below, along with the corresponding coefficient estimates.

## lambda with minimum CV error/1 - SE  
bestlam <- data.frame(min = cv\_out$lambda.min,  
 one\_se = cv\_out$lambda.1se)  
bestlam

min one\_se  
1 0.01629751 0.2656088

|  |
| --- |
| Predictors arranged by absolute values of their estimated coefficients using 1-SE rule from a lasso fit. |

## ## Refit lasso regression  
# The cv\_out object already has the full data fit  
# for each lambda  
lasso\_min = predict(cv\_out$glmnet.fit,  
 type = "coefficients",  
 s = bestlam$min)  
lasso\_1se = predict(cv\_out$glmnet.fit,  
 type = "coefficients",  
 s = bestlam$one\_se)  
# Least squares  
ols <- coef(lm(medv ~ model\_mat))  
betahat\_lasso <- cbind(lasso\_min,  
 lasso\_1se,  
 ols)  
colnames(betahat\_lasso) <- c("min", "1se", "ols")

betahat\_lasso

13 x 3 sparse Matrix of class "dgCMatrix"  
 min 1se ols  
(Intercept) 22.53280632 22.53280632 22.53280632  
crim -0.99497662 -0.40962916 -1.04412968  
zn 1.01701543 0.16905691 1.09530317  
indus . . 0.09239314  
chas 0.72276924 0.59654483 0.72134140  
nox -2.04720216 -0.98224770 -2.17363599  
rm 2.60394572 2.90572805 2.57025715  
age 0.03024013 . 0.10163739  
dis -3.05650182 -1.33624538 -3.13909506  
rad 2.24135435 . 2.51992023  
tax -1.87984102 -0.02201344 -2.13738455  
ptratio -1.99776592 -1.78956364 -2.02970765  
lstat -3.91044836 -3.84867031 -3.94200236

Notice that the coefficient of indus is exactly set to zero, and is thus excluded from the final model, when we choose by minimizing CV error. The one standard error rule gives a much larger , and thus a sparser fit, excluding indus, age and rad from the final model.

## Elastic net

A generalization of lasso and ridge is *elastic net*,[[31]](#footnote-188) which minimizes

for and . Note that lasso and ridge regressions are special cases of elastic net for and , respectively.[[32]](#footnote-189) Zhou and Hastie (2005) suggests that elastic net deals with correlated predictors more effectively than lasso or ridge. The ridge penalty tends to shrink coefficients of correlated variables towards each other, while lasso tends to pick one predictor to be kept in the model while ignoring the rest.[[33]](#footnote-190) The elastic net penalty is a compromise between these two phenomena. The first term the the penalty encourages the correlated features to be averaged, while the second penalty term encourages sparsity in the estimated coefficients of the averaged features.

Elastic net often finds application in genomics (high-dimensional problems) where , and predictors (genes) are often have high correlation among them.

As usual, we need to tune both and in this case. We can use glmnet() to fit elastic net as well.

## Other variable selection methods

There are *many* other variable selection models in literature, including several variations of lasso, such as

* *adaptive lasso*:[[34]](#footnote-192) for estimation with less bias than ordinary lasso. It requires an initial estimate of the coefficients. The penalty term for each coefficient is then inversely weighted by the corresponding initial estimates. We can use the *penalty.factor* argument in glmnet() to do so.
* *group lasso*:[[35]](#footnote-193) for variable selection in groups of variables. For example, we might have a categorical variable with more than two levels. In variable selection, we might exclude/include all the dummy variable together. We can use R package grpreg for fitting group lasso.
* *fused lasso*:[[36]](#footnote-194) does variable selection when the predictors have a natural ordering. For example, the predictors can be genes ordered by their chromosome location. Another example is when predictor is a function of time (functional data or time series). We can use the genlasso package here.
* *Smoothly clipped absolute deviations (SCAD)*[[37]](#footnote-195) and *Minimax concave penalty (MCP)*: produce sparse set of solution and approximately unbiased coefficients for large coefficients. Both methods are available in the ncvreg package.

There are many other methods available in literature. Readers are encouraged to explore according to their needs.

# Dimension Reduction Methods

The variable selection and shrinkage methods discussed so far attempts to reduce model variance in two ways: by reducing number of variables in the model (subset selection, lasso) and by shrinking regression coefficients toward zero (ridge, lasso). Another method to control model variance is to transform the original predictors to obtain new ones, and use them as covariates in the regression model. Typically, the number of new variables are less than the number of the original predictors. Thus these methods are called *dimension reduction* techniques.

Suppose our original predictors are . A typical dimension reduction method has two steps:

1. Create new predictors by transforming/combining the original predictors. Usually we choose , and thus reducing the dimension of the problem.
2. Fit the regression model with the new predictors:

Depending on how we construct the new predictors gives rise to different dimension reduction techniques.

In this section, we will discuss dimension reduction in the context of building linear regression models. We will discuss dimension reduction methods as a part of unsupervised learning in a later chapter.

## Principal Components Regression

Principal components regression uses *Principal Components Analysis (PCA)* to derive new features from the original predictors. For now, we will only briefly discuss PCA – it will be covered in a future chapter.

For simplicity of the following discussion, we will henceforth assume that each predictor variable has been centered.

The objective of PCA is to condense the information that is present in the original set of variables via linear combinations of the variables while losing as little information as possible. Suppose we have a predictors $\X\_i = (X\_{i1},\ldots X\_{ip})^T$. The main goal of PCA is to identify *linear combinations* of the form

that explain most of the variability in the data.[[38]](#footnote-198) Typically we choose , and the new variables, , are ordered according to their importance. Specifically, is designed to capture the most variability in the original variables by any linear combination – this is called the *first principal component (PC)*. Then , the *second PC*, captures the most of the *remaining* variability while being *uncorrelated* to . We continue until we have the -th PC . In the end, we hope that the first few PCs, , will capture most of the variability in the original predictors.

Let us look at the Boston data for a demonstration. In R we can use the function prcomp() to perform PCA. Here we can approximate *total variation* in the original data as the sum of the variances of each predictors.

# Extract only predictors and center them  
X <- scale(Boston[, -13],   
 center = TRUE, scale = FALSE)  
dim(X)

[1] 506 12

# Total variation  
TV = sum(apply(X, 2, var))  
TV

[1] 29998.61

Before proceeding, let us check variances of individual predictors.

apply(X, 2, var)

crim zn indus chas nox rm   
7.398658e+01 5.439368e+02 4.706444e+01 6.451297e-02 1.342764e-02 4.936709e-01   
 age dis rad tax ptratio lstat   
7.923584e+02 4.434015e+00 7.581637e+01 2.840476e+04 4.686989e+00 5.099476e+01

Here we see an obvious problem – the variables are not comparable in terms of their variability. For example, the variable tax has a variance 2.8404759^{4} while lstat has variance 50.9947595. So majority of the total variation is due to tax. In such a case of imbalance, tax will overshadow all other variables. This may not be because tax is the only important variable here, but it is an issue of measurement unit/scale. For example, if we multiply lstat by , it does not make lstat any more important than it originally was, but its variance will be inflated by a factor of making lstat dominant over the rest of the predictors. To avoid this issue, we will standardize each predictor. [[39]](#footnote-199) Since now every predictor will have variance one, total variation is simply the number of predictor in the data.

# Standardized predictors  
Xstd <- scale(X, center = TRUE, scale = TRUE)  
# TV  
TV = ncol(Xstd)  
TV

[1] 12

Now we perform PCA of the predictors. After performing PCA, we will have 12 PCs (linear combinations of the original predictors in $\X$). We can access the PCs in the *$x* component from the prcomp() output.

# PCA  
pc\_out <- prcomp(Xstd)  
names(pc\_out)

[1] "sdev" "rotation" "center" "scale" "x"

# PCs  
Z <- pc\_out$x  
dim(Z)

[1] 506 12

Each column of contain one PC – the first column if for PC1, the second for PC2 and so on. First we note that the combined variation in is exactly the same as total variation in the original predictors.

sum(apply(Z, 2, var))

[1] 12

Thus the ratio of variance of the 1st PC (first column of Z) to the total variation quantifies how much of the total variation is captured by the 1st PC. We can do similar calculations for each of the PCs.

# Proportion of TV captured by PC1  
var(Z[,1])/TV

[1] 0.4922571

The PCs are ordered by their variance. By construction, PCs are uncorrelated. So total variation captured by the first few PCs is simply the sum of their individual variances. We can define proportion of variation similarly.

# Cumulative proportion of TV captured by successive PCS  
cumsum(apply(Z, 2, var)) / TV

PC1 PC2 PC3 PC4 PC5 PC6 PC7 PC8   
0.4922571 0.6089290 0.7073204 0.7785239 0.8453863 0.8900683 0.9230771 0.9462417   
 PC9 PC10 PC11 PC12   
0.9650053 0.9805225 0.9947065 1.0000000

In the example above we can see that the first three PCs together explain 70.732 percent of total variation.

We can use the summary() function to see the perfromance of PCA.

summary(pc\_out)

Importance of components:  
 PC1 PC2 PC3 PC4 PC5 PC6 PC7  
Standard deviation 2.4304 1.1832 1.08660 0.9244 0.89574 0.73225 0.62937  
Proportion of Variance 0.4923 0.1167 0.09839 0.0712 0.06686 0.04468 0.03301  
Cumulative Proportion 0.4923 0.6089 0.70732 0.7785 0.84539 0.89007 0.92308  
 PC8 PC9 PC10 PC11 PC12  
Standard deviation 0.52723 0.47451 0.43152 0.41256 0.25204  
Proportion of Variance 0.02316 0.01876 0.01552 0.01418 0.00529  
Cumulative Proportion 0.94624 0.96501 0.98052 0.99471 1.00000

Note that we need all the 12 PCs to capture of , but doing so will not perform dimension reduction. Thus we will have discard the last few PCs and in the process sacrifice some of the variation in the original data. For example, if we are willing to sacrifice of TV (i.e., capture of TV), we will only need 6 PCs. *In principal component regression, we will treat the number of PCs to retain as a tuning parameter.*

Let us not briefly look at the *loadings* for the PCs.[[40]](#footnote-200)

loadings <- pc\_out$rotation  
# PC1 loadings  
round(loadings[,1], 2)

crim zn indus chas nox rm age dis rad tax   
 0.25 -0.27 0.35 0.01 0.35 -0.20 0.32 -0.33 0.32 0.34   
ptratio lstat   
 0.21 0.32

It seems PC1 has two groups of variables, (zn, rm and dis) vs. the rest of the variables excluding chas, with loadings with opposite signs but roughly similar magnitude. Investigation of correlation plot (Figure ) of the predictors gives insight about PC1 loadings. We can see there are two groups of variables that have positive correlation within each group, but have negative correlation between the groups. PC1 essentially quantifies this pattern.

|  |
| --- |
| Correlation plot of Boston data. |

From a geometric point of view, PCA attempts to find the *directions along which most of the variability is present*. Let us consider the simple case with number of variables . Thus for PC1 we need to determine loading so that variance of is maximized. The condition on the loadings is

This is the equation of a circle, centered at zero, with radius one. So we only need to look at points that are on the perimeter of the circle. This is what we mean by *direction*; see Figure .

|  |
| --- |
| First PC direction. |

Thus, given a data scatterplot, the 1st PC points to the direction along with most of the variation lies. In Figure , the grey points represent a data scatter. PCA first places a circle of unit length at the center of the data (the black circle in the plot) and finds the direction with the most variation (the red arrow). The direction orthogonal to PC1 containing the second largest amount of variation is PC2 (the blue arrow).

|  |
| --- |
| Geometry of PCA in two and three dimensions (left and right panels, respectively). |

Let us now consider the case with three variables, . In this case, the loadings are and the constraint becomes

This is the equation of a sphere, centered at zero, with radius one. Thus we only need to look at points that are on the *surface of the sphere*.

Now consider a data scatter in three dimensions (gray points in Figure , right panel). We first place a sphere of unit radius at the center of the data (the light-blue sphere). Then the first PC points to the direction (represented by the vector on the surface of the sphere) with the most variation (the red arrow). The second PC is the direction orthogonal to the first PC containing the second largest amount of variation. The third PC is the direction orthogonal to both the first and second PCs.

Note that any direction represented by the vector $\ab$ is also represented by $-\ab$ (just like “x-axis” corresponds to both positive and negative directions). Thus if $\ab$ is a PC then so is $-\ab$. In other words, if is a PC, then so is . Thus it is not advisable to interpret the loadings as they are (since the sign is unidentifiable) – we need to interpret them *relative to* other loadings. For example, we can say crim has opposite relationship to PC1 compared to zn.

Now that we have constructed the PCs, we can choose the first PCs, , and build a regression model with the PCs as predictors. Here we are assuming that the direction that the original predictors, show most variation are in fact the directions associated with the response.[[41]](#footnote-210)

If the assumption above holds true, then using PCR with as predictors will give a better result than using all the original predictors. PCR may also help mitigating overfitting.

The number of retained PCs, , is considered to be a tuning parameter and can be chosen by cross-validation (or other data splitting methods). Once the optimal is chosen, we fit the model to the full data with the chosen to obtain the final model.

In R, we can use the pcr() function in the pls package.[[42]](#footnote-211) Note the usage of arguments center = TRUE and scale = TRUE.

library(pls)  
set.seed(1001)  
pcr\_lm <- pcr(medv ~ .,   
 data = Boston,  
 center = TRUE, scale = TRUE,  
 validation = "CV")

When using pcr(), we do not need to explicitly obtain the PCs – it is automatically done by pcr(). Here we use the original Boston data, and use the scale and center arguments to standardize. The validation argument specifies the method to choose .

summary(pcr\_lm)

Data: X dimension: 506 12   
 Y dimension: 506 1  
Fit method: svdpc  
Number of components considered: 12  
  
VALIDATION: RMSEP  
Cross-validated using 10 random segments.  
 (Intercept) 1 comps 2 comps 3 comps 4 comps 5 comps 6 comps  
CV 9.206 7.322 6.562 5.566 5.363 5.208 5.224  
adjCV 9.206 7.321 6.556 5.562 5.353 5.201 5.217  
 7 comps 8 comps 9 comps 10 comps 11 comps 12 comps  
CV 5.197 5.199 5.199 5.179 4.996 4.930  
adjCV 5.190 5.193 5.190 5.171 4.986 4.919  
  
TRAINING: % variance explained  
 1 comps 2 comps 3 comps 4 comps 5 comps 6 comps 7 comps 8 comps  
X 49.23 60.89 70.73 77.85 84.54 89.01 92.31 94.62  
medv 36.96 50.50 64.15 67.32 69.04 69.06 69.58 69.61  
 9 comps 10 comps 11 comps 12 comps  
X 96.50 98.05 99.47 100.00  
medv 70.17 70.53 72.59 73.43

The cross-validation results suggest that the lowest RMSE corresponds to using all 12 PCs. In other words, in this example, PCR did not provide any benefit.

|  |
| --- |
| Cross-validation error with one SE error bars. |

Let us now investigate the one standard error rule in this situation. Specifically, we can choose a smaller model whose test error is within one standard error of the minimum test error. For computational ease, let us refit PCR and perform cross-validation using caret.

set.seed(1001)  
model <- train(medv ~ .,   
 data = Boston,  
 method = "pcr",  
 trControl = trainControl("cv", number = 10),  
 tuneLength = 12,  
 preProcess = c("center", "scale")  
 )

model$results

ncomp RMSE Rsquared MAE RMSESD RsquaredSD MAESD  
1 1 7.188782 0.3962757 5.026684 1.3771210 0.13809005 0.8096569  
2 2 6.414383 0.5148207 4.640147 1.1904223 0.10504920 0.6549203  
3 3 5.455026 0.6492901 3.856546 1.1050287 0.08938897 0.5756135  
4 4 5.177018 0.6829000 3.617265 1.2288044 0.11653420 0.6809517  
5 5 5.084529 0.6955340 3.494738 1.1440713 0.10635726 0.6327532  
6 6 5.100443 0.6938870 3.503544 1.1420132 0.10620278 0.6499091  
7 7 5.088361 0.6962521 3.515851 1.0707799 0.09757015 0.5615676  
8 8 5.103823 0.6946868 3.521178 1.0768574 0.09828127 0.5693120  
9 9 5.163096 0.6888286 3.599038 1.0223223 0.09285216 0.5486695  
10 10 5.130536 0.6925015 3.560661 1.0438463 0.09533109 0.5536748  
11 11 4.944944 0.7164652 3.530450 0.8887682 0.07539532 0.4800970

Since we are using 10-fold CV, the standard error of the estimate of the test error (average of the 10 test errors) is simply the standard deviation divided by square-root of number of folds.[[43]](#footnote-215)

SE <- model$results$RMSESD/sqrt(10)  
round(SE, 2)

[1] 0.44 0.38 0.35 0.39 0.36 0.36 0.34 0.34 0.32 0.33 0.28

Thus we can plot the estimated test errors and error bars representing plus/minus one standard error, see Figure .

Now we see that a model with 4 PCs can be chosen with the one standard error rule. From the PCA output shown earlier, first 4 PCs explain 77.85 percent of total variation in the original data. Finally, we fit the model chosen number of PCs.

pcr\_final <- pcr(medv ~ .,  
 data = Boston,  
 center = TRUE, scale = TRUE,  
 ncomp = 4, validation = "none")  
summary(pcr\_final)

Data: X dimension: 506 12   
 Y dimension: 506 1  
Fit method: svdpc  
Number of components considered: 4  
TRAINING: % variance explained  
 1 comps 2 comps 3 comps 4 comps  
X 49.23 60.89 70.73 77.85  
medv 36.96 50.50 64.15 67.32

While PCR performs dimension reduction, it does *not* perform variable selection since each PC can be a combination of all the original variables. For example, in our final model with 4 leading PCs, the model is

Since , we can write the model above in terms of the original variables as

Therefore, PCR includes all the original variables in the final model. In our example, the coefficients for the standardized original coefficients can be obtained follows.

coef(pcr\_final)

, , 4 comps  
  
 medv  
crim -0.3678349  
zn 0.6767729  
indus -0.6154494  
chas 0.6128129  
nox 0.2643513  
rm 3.6999912  
age 0.1472266  
dis -0.5926663  
rad -0.2078194  
tax -0.4315474  
ptratio -2.3749697  
lstat -2.2448475

Note that none of the original variables has zero coefficients.

## Partial Least Squares

In PCR, we are assuming that the direction that the original predictors, show most variation are in fact the directions associated with the response. Such an assumption need to hold true since the PC directions are computed in an unsupervised way.

In contrast, partial least squares (PLS)[[44]](#footnote-217) is a *supervised* approach, that is, PLS determines the linear combinations of the original predictors by making use of the response. Roughly speaking, the PLS approach attempts to find directions that help explain both the response and the predictors.

Recall, we are still operating under the assumption that the predictors have been standardized. PLS begins by performing a *simple linear regression* of on the -th original predictor, , for each . The resulting estimates of slopes are denoted as , respectively. Then the first PLS component is constructed as

Thus the first PLS component places the highest weight on the variables that are most strongly related to the response.

To construct the second PLS component, we regress each predictor variable on the first PLS component, and take the residuals. We can view these residuals as the remaining information that has not been captured by the first PLS component. The the second PLS component is computed in the same manner as before: where if estimated regression coefficient of from the simple linear regression of the residuals (obtained above) on . We continue this process until we have all the PLS components. As in PCR, we take the leading PLS components. A multiple linear regression is then fitted with as response and the PLS components as predictors.

We can use the plsr() function in pls package, or use caret with mehod = "pls". The number of PLS components, can be chosen using data splitting methods, such as CV.

set.seed(1001)  
model <- train(medv ~ .,   
 data = Boston,  
 preProcess = c("center", "scale"),  
 method = "pls",  
 trControl = trainControl("cv", number = 10),  
 tuneLength = 12  
 )  
model

Partial Least Squares   
  
506 samples  
 12 predictor  
  
Pre-processing: centered (12), scaled (12)   
Resampling: Cross-Validated (10 fold)   
Summary of sample sizes: 455, 455, 454, 456, 456, 456, ...   
Resampling results across tuning parameters:  
  
 ncomp RMSE Rsquared MAE   
 1 6.416461 0.5187161 4.466546  
 2 5.037699 0.7007280 3.491181  
 3 4.976713 0.7103322 3.445529  
 4 4.953041 0.7159024 3.511223  
 5 4.912530 0.7206881 3.512850  
 6 4.877108 0.7241979 3.490279  
 7 4.868411 0.7255914 3.485247  
 8 4.859391 0.7262403 3.478366  
 9 4.859906 0.7262037 3.476025  
 10 4.862116 0.7260036 3.477238  
 11 4.861939 0.7260194 3.477140  
  
RMSE was used to select the optimal model using the smallest value.  
The final value used for the model was ncomp = 8.

|  |
| --- |
| Cross-validation error with one SE error bars. |

We can plot the estimated test errors and error bars representing plus/minus one standard error as we did in PCR – see Figure . Using minimum test error, we can use 8 PLS components. Using one standard error rule, it seems two PLS components are sufficient.

We can finally fit the PLS regression model on the full data using the chosen number of PLS components. Using 8 PLS components, the final fit is shown below.

pls\_final <- plsr(medv ~ .,  
 data = Boston,  
 center = TRUE, scale = TRUE,  
 ncomp = 8)  
summary(pls\_final)

Data: X dimension: 506 12   
 Y dimension: 506 1  
Fit method: kernelpls  
Number of components considered: 8  
TRAINING: % variance explained  
 1 comps 2 comps 3 comps 4 comps 5 comps 6 comps 7 comps 8 comps  
X 47.81 59.71 67.51 72.82 79.09 82.22 86.00 89.95  
medv 49.84 69.90 71.58 72.75 73.15 73.30 73.38 73.42

We can extract the values of PLS components (scores), that is, values using the scores() function. The weights (loadings) of the original variables for each PLS components can be extracted using loadings() function.

pls\_scores <- scores(pls\_final)  
load <- loadings(pls\_final)

It can be shown that PLS computes directions that have high variance and have high correlation with the response. In contrast, PCA seeks directions only with high variance.[[45]](#footnote-221) [[46]](#footnote-222) In practice PLS often produces performance similar to ridge regression or PCR. While the supervised dimension reduction of PLS can reduce bias, it also has the potential to increase variance.

# High-dimensional data

So far, all the methods we discussed assume that the number of predictors () is (much) less than the sample size (). The performance of these methods deteriorate as gets closer or exceed . Data sets containing more features than observations (or sometimes number of features slightly smaller than ) are often referred to as *high-dimensional*. In many fields, such as genomics and bioinformatics, such high-dimensional data are common. For example, in genomics we measure *single nucleotide polymorphisms (SNPs)*[[47]](#footnote-225) and investigate their association with an outcome of interest. Typically, the number of SNPs are in hundred of thousands, but sample size is in hundreds.

When we have , usual least squares regression should not be performed. This is because as , the model matrix will not have full column rank, and as such least squares does not provide unique solutions. Furthermore, training set measures such as and will keep getting better and better as we add more predictors to the model *regardless whether the predictors are actually associated with the response*. Suppose we have predictors. When (or if intercept is not in the model), least squares gives a perfect fit with zero residuals ( and ). However, such a model will perform extremely poorly in a test set due to very high model variance. Figure further illustrates the risk of carelessly applying least squares when the number of features is large.

|  |
| --- |
| Risk of carelessly applying least squares when the number of features is high. Data were simulated with n = 20 observations, and regression was performed with between 1 and 20 features, each of which was completely unrelated to the response. |

In fact, the model evaluation approaches that do not require a test set (AIC, BIC, adjusted ), are also not appropriate for in the high-dimensional setting due to instability of estimation of and RSS, both of which will be zero when . Thus we need alternative methods in this situation.

## Regression in high-dimensions

We can still apply *dimension reduction approaches* such as forward stepwise selection[[48]](#footnote-229), ridge regression, the lasso, and principal components regression. These methods avoid overfitting data using a less flexible model.

|  |
| --- |
| The lasso was performed with n = 100 observations and three values of p, the number of features. Of the features, 20 were associated with the response. The boxplots show the test MSEs that result using three different values of the tuning parameter . For ease of interpretation, rather than reporting , the degrees of freedom are reported; for the lasso this turns out to be simply the number of estimated non-zero coefficients. When , the lowest test MSE was obtained with the smallest amount of regularization. When , the lowest test MSE was achieved when there is a substantial amount of regularization. When the lasso performed poorly regardless of the amount of regularization, due to the fact that only of the features truly are associated with the outcome. |

Figure illustrates the performance of the lasso in a simple simulated example (figure taken from *Introduction to Statistical Learning*). The *degrees of freedom* used in the plot is simply the number of non-zero coefficients in the lasso model. Large degrees of freedom indicate a more flexible model. The sample size uses the simulation is . It is evident that *test error increases as the the number of predictors increases*, unless the additional features are truly associated with the response. This phenomenon is called the *curse of dimensionality*.

In general, test MSE will decrease by adding predictors that are truly associated with the response. Adding noise predictors that are not related to the response at all will lead to an increase of test MSE. This is because adding such noise predictors increases dimensionality of the problem and results in overfitting.

## Interpreting Results in High Dimensions

Another issue in high-dimensional problem is multicollinearity, that is, when one predictor can be expressed as a linear combination of the others. When , the predictors will *always* have multicollinearrity – any predictor can be written as a linear combination of the others. This implies that we can not identify the best coefficient in the regression model. At most, we can hope to assign large regression coefficients to variables that are correlated with the variables that truly are predictive of the outcome.

We should also be careful in reporting measures of model fit. We quote the following paragraph from Chapter 6.4 of *Introduction to Statistical Learning*.

We have seen that when , it is easy to obtain a useless model that has zero residuals. Therefore, one should never use sum of squared errors, p-values, statistics, or other traditional measures of model fit on the training data as evidence of a good model fit in the high-dimensional setting.

It is important to instead report results on an independent test set, or cross-validation errors. For instance, the MSE or on an independent test set is a valid measure of model fit, but the MSE on the training set certainly is not.

1. Generalized inverse of is a matrix such that . Although there are other definitions used by various authors. [↑](#footnote-ref-37)
2. A matrix can not have full column rank if it has more rows than columns. [↑](#footnote-ref-38)
3. If we need to remove the intercept, we need to specify “~ -1 + lstat” [↑](#footnote-ref-50)
4. Use names(simple\_ols) to see all the components of the output. Of special interest are: “coefficients” (estimate parameters), “residuals”, and “fitted.values” (predicted response of the training set). [↑](#footnote-ref-51)
5. Here the term I(lstat^2) tells the formula that the square of lstat should be computed as is. [↑](#footnote-ref-55)
6. Alternatively, we can explicitly specify the interaction term in the formula: “lstat + age + lstat:age” [↑](#footnote-ref-59)
7. There are ways to relax these assumptions. For example, if we have large sample size, we might relax the normality assumption under certain conditions on . [↑](#footnote-ref-65)
8. Here corresponds to the intercept . [↑](#footnote-ref-66)
9. Here the vcov() function produces the matrix .The function diag() then extracts the diagonal elements of the matrix, and then we we take the square root by using the sqrt() function. [↑](#footnote-ref-67)
10. Probability computed over repeated sampling. [↑](#footnote-ref-69)
11. Note that we are stating that probability of the true value of falls between and is . This is clearly a wring statement since the probability is either 0 or 1. [↑](#footnote-ref-73)
12. In general, to test , for any fixed value of , we use the test statistic [↑](#footnote-ref-75)
13. In general, a level test would correspond to a confidence interval. [↑](#footnote-ref-79)
14. Especially when there are large number of predictors (large ). [↑](#footnote-ref-81)
15. Note that TSS is proportional to the sample variance of the ’s. [↑](#footnote-ref-84)
16. See ?plot.lm() for details. [↑](#footnote-ref-94)
17. A rule of thumb could be that possible outliers are observations with studentized residuals more than 3 in absolute value. [↑](#footnote-ref-104)
18. See ?predict.lm for details. [↑](#footnote-ref-117)
19. Recall that for any event , we define the *indicator function* if is true, 0 otherwise. [↑](#footnote-ref-125)
20. See ?ames\_raw after loading AmesHousing package. [↑](#footnote-ref-126)
21. See Introduction to Statistical Learning, Chapter 6.1.3. [↑](#footnote-ref-128)
22. Note that the intercept is not penalized. [↑](#footnote-ref-153)
23. The idea of using the sum-of-squares of the parameters as penalty is also used in neural networks – it is known as . [↑](#footnote-ref-154)
24. Here denotes the identity matrix: a diagonal matrix with all diagonal elements being 1. [↑](#footnote-ref-155)
25. This was the original motivation behind development of ridge regression, see Hoerl and Kennard (1970), Ridge Regression: Biased Estimation for Nonorthogonal Problems, Technometrics, 12, 55 – 67. [↑](#footnote-ref-156)
26. This is the reason we call least squares estimators *scale equivariant*. [↑](#footnote-ref-160)
27. See ?glmnet for more details. [↑](#footnote-ref-161)
28. glmnet() has a default way to set values as well if we do not specify manually. [↑](#footnote-ref-162)
29. See ?glmne.cv() for details. We can use nfolds argument to specify number of folds while using CV. [↑](#footnote-ref-166)
30. Computing the lasso solution is a . Efficient algorithms are available for computing the entire path of solutions as is varied. These algorithms have the same computational cost as for ridge regression. Interested readers should see for details. [↑](#footnote-ref-174)
31. Zou H, Hastie T (2005). Regularization and Variable Selection via the Elastic Net. Journal of the Royal Statistical Society, Series B, 67(2), 301–320. [↑](#footnote-ref-188)
32. This is the formulation glmnet() uses with the alpha argument. [↑](#footnote-ref-189)
33. See for more discussion. [↑](#footnote-ref-190)
34. Zou, H (2012). The Adaptive Lasso and Its Oracle Properties, JASA, 101, 1418 - 1429 [↑](#footnote-ref-192)
35. Yuan, M. & Lin, Y. (2007), Model selection and estimation in regression with grouped variables, Journal of the Royal Statistical Society, Series B 68(1), 49 - 67 [↑](#footnote-ref-193)
36. Tibshirani, R., Saunders, M., Rosset, S., Zhu, J. and Knight, K. (2005), “Sparsity and smoothness via the fused lasso”, Journal of the Royal Statistics Society: Series B 67(1), 91 - 108. [↑](#footnote-ref-194)
37. Fan J and Li R. (2001). Variable Selection via Nonconcave Penalized Likelihood and its Oracle Properties. Journal of American Statistical Association, 96:1348 - 1360. [↑](#footnote-ref-195)
38. Mathematically, we need to *normalize* the weights, that is, we ensure that . [↑](#footnote-ref-198)
39. This is my general recommendation when performing PCA. [↑](#footnote-ref-199)
40. We will discuss more about interpreting the loadings in a later chapter. [↑](#footnote-ref-200)
41. There is no assurance such an assumption actually holds. [↑](#footnote-ref-210)
42. Other packages such as caret can also so this. [↑](#footnote-ref-211)
43. Recall, for a random sample , standard error of sample mean is [↑](#footnote-ref-215)
44. Originally, Herman Wold developed the nonlinear iterative partial least squares (NIPALS) algorithm (Wold 1966, 1982) algorithm for nonlinear models. Later, Wold et al. (1983) adapted the NIPALS method for regression setting with correlated predictors – this adaptation was named PLS. [↑](#footnote-ref-217)
45. Stone M, Brooks R (1990). Continuum Regression: Cross-validated Sequentially Constructed Prediction Embracing Ordinary Least Squares, Partial Least Squares, and Principal Component Regression. Journal of the Royal Statistical Society, Series B, 52, 237 - 269. [↑](#footnote-ref-221)
46. Frank, I.E. and Friedman, J.H. (1993) An Statistical View of Some Chemometrics Regression Tools. Technometrics, 35, 109 - 135. [↑](#footnote-ref-222)
47. These are individual DNA mutations that are relatively common in the population [↑](#footnote-ref-225)
48. Backward selection can not be used here since we can not a fit the full model with all the predictors. [↑](#footnote-ref-229)