# *Linear Regression*

*Arnab Maity*

*NCSU Statistics ~ 5240 SAS Hall ~ amaity[at]ncsu.edu*

*Contents*

## *Introduction*

Linear regression is a simple supervised learning tool for modeling a quantitative response. It 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 though of generalizations of linear regression model. Practically, in real data, often we see relationships that *locally* linear. In other cases, even if the original relationship is not linear, one may transform the response and predictors (e.g., using a *log*-transform) to get linearity.

A linear regression model has the form

$$Y_i = \beta_0 + X_{i1}\beta_1 + X_{i2}\beta_2 + \ldots + X_{ip}\beta_p + \epsilon_i,$$

where $Y_i$ is a quantitative response, $X_{i1}, \ldots, X_{ip}$ are predictor variables, and $\epsilon_i$ is unobserved random error. Here $\beta_0$ is th e *intercept*, the coefficient $\beta_j, j = 1, \ldots, p$ are coefficients associated with predictor $X_{ij}$. The value of $\beta_j$ indicate the strength, and direction, of the *linear* relationship between $X_{ij}$ and $Y_i$.[1] Typically, we assume that the errors have mean zero, that is $E(\epsilon_i) = 0$. Thus we can write the mean response as

[1] A special case is the *simple linear regression model*, which has only one predictor $X$.

$$E(Y_i|X_{i1}, \ldots, X_{ip}) = \beta_0 + X_{i1}\beta_1 + X_{i2}\beta_2 + \ldots + X_{ip}\beta_p.$$

Such a representation also gives us an interpretation of the $\beta$ parameters: $\beta_j$ is the *rate of change* in the mean response due to one unit increase in the $j$-th predictor, while keeping the other variable fixed. We call such regression models as *linear models*.

**Linear models**

We call the model described above *a linear model* because $E(Y|X)$ is *linear in parameters*, that is, linear in $\beta$.

An implication of the definition above is that we can have nonlinear terms of $X$, but the model would be still a linear model. For example, the so-called polynomial regression,

$$E(Y_i|X_i) = \beta_0 + X_i\beta_1 + \ldots + X_i^d\beta_d,$$

fractional polynomial regression[2]

[2] Royston P, Sauerbrei W (2004) A new approach to modelling interactions between treatment and continuous covariates in clinical trials by using fractional polynomials. Stat Med 23:2509–2525

$$E(Y_i|X_i) = \beta_0 + X_i^{r_1}\beta_1 + \ldots + X_i^{r_d}\beta_d,$$

where the powers $r_1, \ldots, r_d$ are chosen from $\{-2, -1, -0.5, 0.5, 1, 2, 3\}$, and models with interaction such as

$$E(Y_i|X_{i1}, X_{i2}) = \beta_0 + X_{i1}\beta_1 + X_{i2}\beta_2 + X_{i1}X_{i2}\beta_3,$$

are all linear models.[3] Thus linear models do not just capture linear effects of $X$, they can capture nonlinear functions of $X$ as well. Figure 1 shows a few examples functions that can be captured by appropriate linear models.

A common estimation procedure for the regression coefficients is the *least squares* technique, which we will learn about in the next section. We denote the estimated coefficients as $\widehat{\beta}_j$, and the in-sample predictions are

$$\widehat{Y}_i = \widehat{\beta}_0 + X_{i1}\widehat{\beta}_1 + X_{i2}\widehat{\beta}_2 + \ldots + X_{ip}\widehat{\beta}_p.$$

In general, for new data points $X_{new,1}, \ldots, X_{new,p}$ the corresponding prediction is

$$\widehat{Y}_{new} = \widehat{\beta}_0 + X_{new,1}\widehat{\beta}_1 + X_{new,2}\widehat{\beta}_2 + \ldots + X_{new,p}\widehat{\beta}_p.$$

The difference between the observed and predicted values are called *residuals*,

$$e_i = Y_i - \widehat{Y}_i.$$

We define the *residual sum of squares*, also known as *sum-of-squared errors (SSE)* as

$$RSS = \sum_i e_i^2 = \sum_i (Y_i - \widehat{Y}_i)^2.$$

We see that MSE is simply RSS divided by the sample size. Thus we can compute test/training MSE form the corresponding RSS. Let us now move onto estimation procedure for linear models.

[3] In contrast, a model such as $E(Y|X) = e^{\beta_0 + X\beta_1}$ is a *nonlinear* regression model.

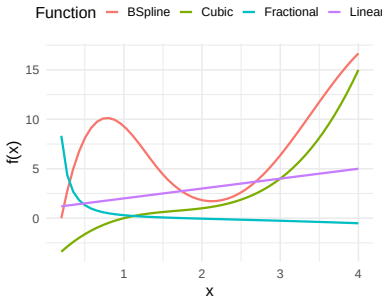



Figure 1: Examples of functions that can be captured by appropriate linear models.

## *Least Squares Estimation*

Recall that our linear regression model has the form:

$$Y_i = \beta_0 + X_{i1}\beta_1 + X_{i2}\beta_2 + \ldots + X_{ip}\beta_p + \epsilon_i.$$

The *ordinary least squares (OLS)* procedure estimates $\beta_j, j = 0, \ldots, p$ by minimizing the sum-of-squares

$$\sum_{i=1}^{n} (Y_i - \beta_0 - X_{i1}\beta_1 - \ldots - X_{ip}\beta_p)^2,$$

with respect to $\beta_j$'s. The resulting estimates are called the ordinary least squares estimates of $\beta_j$.

In the special case with only one predictor, that is, the *simple linear regression* model

$$Y_i = \beta_0 + X_i\beta_1 + \epsilon_i,$$

the OLS estimators of the regression coefficients are

$$\widehat{\beta}_1 = \frac{\sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{\sum_{i=1}^n (X_i - \bar{X})^2}, \text{ and } \widehat{\beta}_0 = \bar{Y} - \bar{X}\widehat{\beta}_1.$$

Figure 2 shows the geometry of the OLS procedure when there is only one predictor. In Figure 2, the response is `Sales` and the predictor is `TV`.

For the general regression model with $p$ predictors, writing closed form expression is easier in matrix from. We can convert the original regression model in matrix form as

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon},$$

where $\mathbf{Y} = (Y_1, \ldots, Y_n)^T$ is the column vector of responses, $\mathbf{X}$ is $n \times (p+1)$ matrix, $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)^T$ is the column vector of regression coefficients, and $\boldsymbol{\epsilon} = (\epsilon_1, \ldots, \beta_n)^T$ is the column vector of errors. We call $\mathbf{X}$ the *model matrix*. The first column of $\mathbf{X}$ has all elements equal to 1 (corresponding to the intercept). For the remaining part of $\mathbf{X}$, each row corresponds to an unit/individual and each column corresponds to a predictor. Thus for the special case of simple linear regression with one predictor $X$, the model matrix is of size $n \times 2$,

$$\mathbf{X} = \begin{bmatrix} 1 & X_1 \\ \vdots & \vdots \\ 1 & X_n \end{bmatrix}.$$

In general, we have

$$\mathbf{X} = \begin{bmatrix} 1 & X_{11} & \ldots & X_{1p} \\ \vdots & \vdots & \ldots & \vdots \\ 1 & X_{n1} & \ldots & X_{np} \end{bmatrix}.$$

With the setup above, there is a *unique* minimizer of the sum-of-squares – the estimator of the regression parameter vector is[4]

$$\widehat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}.$$

Figure 3 shows the geometry of the OLS procedure when there are predictors, $E(Y_| X_{i1}, X_{i2}) = \beta_0 + X_{i1}\beta_1 + X_{i2}\beta_2$.

The estimator above depends on the term $(\mathbf{X}^T\mathbf{X})^{-1}$, that is, the inverse of $(\mathbf{X}^T\mathbf{X})$. Such an inverse exists only if $\mathbf{X}$ *has full column-rank*. Equivalently, $\mathbf{X}$ must have the following two properties:

(C1) The sample size $n$ is larger than the number of regression coefficients in the model, $p+1$.

(C2) None of the columns of $\mathbf{X}$ can be written as a weighted sum (called a *linear combination*) of the remaining columns.

If $\mathbf{X}$ violates either of these conditions, then a *unique* least squares estimator does not exist. If $\mathbf{X}$ violates (C2) but not (C1), then we can replace $(\mathbf{X}^T\mathbf{X})^{-1}$ by a *generalized inverse*[5], but there will be many esti-

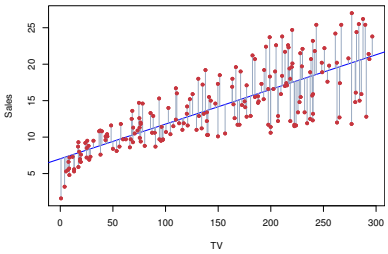



Figure 2: Geometry of the OLS procedure. Each grey line segment represents a residual. The best values of regression coefficients are found by minimizing sum of sqaures of these residuals. Figure taken from *Introdction to Statistical Learning*.

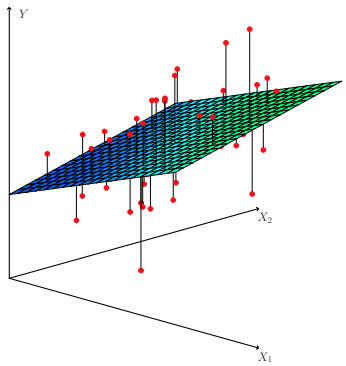



Figure 3: Geometry of the OLS procedure with two preditors with no interaction. Each grey line segment represents a residual. Figure taken from *Introdction to Statistical Learning*.

[4] Here we denote the *inverse* of a square matrix $\mathbf{A}$ by $\mathbf{A}^{-1}$, when it exists.

[5] Generalized inverse of $\mathbf{A}$ is a matrix $\mathbf{G}$ such that $\mathbf{AGA} = \mathbf{A}$. Although there are other definitions used by various authors.


5240 SAS Hall, amaity[at]ncsu.edu

mators 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 $\mathbf{X}$ violates (C1) then (C2) is automatically violated.[6] In that case, one can take a few steps described below.

[6] A matrix can not have full column rank if it has more rows than columns.

- 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.

We will learn the techniques mentioned above in future.

A drawback of least squares is that it is susceptible to influential points, the points that have unduly high impact on the regression process, such as parameter estimates, predicted response etc.

**Influential points**

**Outlier:** A point for which the response ($Y$) is far from the value predicted by the model.

**High leverage point:** A point with unusual value of predictor ($X$), i.e., large/small values compared to the rest of the data.

Figure 4 shows an example of an outlier and its impact of the regression fit.

We could simply remove the outlier from the data set, and/or use a different minimization criterion that are more robust to influential points. For example, instead of minimizing the squared error criteria $\sum_i e_i^2$, one can use the *least absolute deviation (LAD)* criterion,[7]

$$\sum_{i=1}^{n} |e_i|,$$

or *Huber function*, which uses squared residuals when their values are small, but uses absolute value for large residuals (above a certain cutoff). In general, *robust regression* methods are often of interest if data are prone to large outliers or have a heavy tailed distribution.

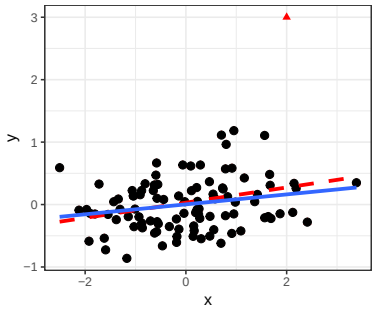



Figure 4: Example of an outlier (red point). Shown are two regression lines: before (red dashed) and after (blue solid) removing the outlier.

[7] Diminishes the effect of outliers by taking absolute value rather than square.

*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 $\mathbf{X}$ 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 $\mathbf{X}$ 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 $\mathbf{X}$ does not depend on the response. The right hand side contains the regression formula. Note that the intercept is automatically included[8] – 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.[9]

```
simple_ols <- lm(medv ~ lstat,
                 data = Boston)
simple_ols$coefficients
```

```
## (Intercept)       lstat
##  34.5538409  -0.9500494
```

Figure 5 shows the least squares fit to the `Boston` data. We can see the estimate intercept, $\widehat{\beta}_0 \approx 34.55$, and the slope $\widehat{\beta}_1 \approx -0.95$. The estimated slope the rate of change in $E(Y|X)$ for each unit increase in $X$. In other words, every 1 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 5, 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,

[8] If we need to remove the intercept, we need to specify "`~ -1 + lstat`"

[9] Use `names(simple_ols)` to see all the components of the output. Of special interest are: "coefficients" (estimate $\beta$ parameters), "residuals", and "fitted.values" (predicted response of the training set).

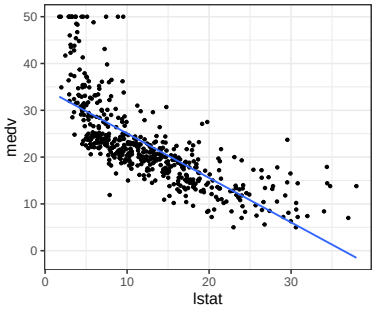



Figure 5: Least squares fit to the simple linear regression model with 'medv' as response and 'lstat' as the predictor.

we can fit a quadratic term of `lstat` as well.[10] Figure 6 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
```

We can fit interaction terms using the `X1*X2` notation. Thus the fomula `Y ~ X1*X2` will include *main effects* of $X_1$ and $X_2$, and the *two-way interaction effect* $X_1X_2$ in the model. For example, we can fit a model with `lstat`, `age` and their interaction as follows.[11]

```
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 $X_1X_2$, we also retain terms $X_1$ and $X_2$. 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

$$Y = \beta_0 + X_1\beta_1 + X_2\beta_2 + X_1X_2\beta_3 + \epsilon.$$

Then for a fixed value of $X_2 = x_2$, one unit increase in $X_1$ corresponds to $\beta_1 + x_2\beta_3$ amount change in $Y$, not just $\beta_1$. Thus the fitted surface will have some curvature. Figure 7 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 $-1.392 + 0.004 * age$ in `medv` on average. On the other hand, one unit increase in `age` corresponds to a change of $-0.001 + 0.004 * lstat$ 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.

[10] Here the term `I(lstat^2)` tells the formula that the square of lstat should be computed as is.

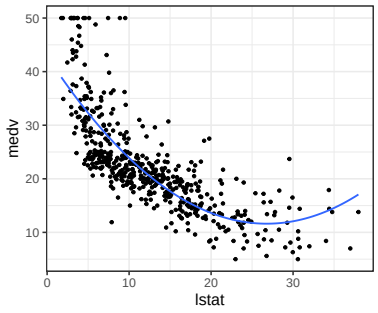



Figure 6: Least squares fit to the simple linear regression model with 'medv' as response and 'lstat' as the predictor.

[11] Alternatively, we can explicitly specify the interaction term in the formula: "`lstat + age + lstat:age`"

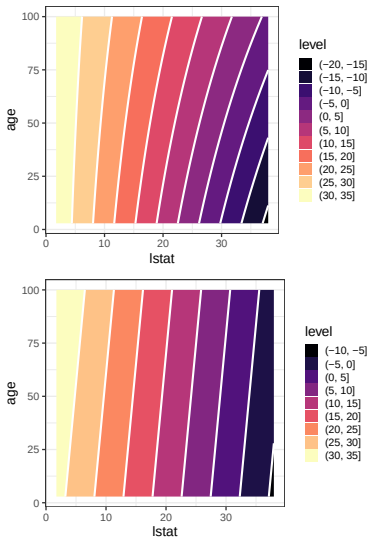



Figure 7: Contour plots of fitted surface in regression without (top) and with (bottom) a two-way interaction term.

To this end, we require additional assumptions on the error terms $\epsilon_i$. For simplicity, we will use the following set of assumptions:[12]

- The errors $\epsilon_i$ are independently and identically distributed as $N(0, \sigma^2)$.
- Errors are independent of the covariates $X_i$

[12] 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 $\mathbf{X}$.

The assumptions above imply that $Y_i|X_i \sim N(\beta_0 + X_i\beta_1, \sigma^2)$.

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

$$SE(\widehat{\beta}_j) = \sqrt{var(\widehat{\beta}_j)}.$$

For a general multiple linear regression model with model matrix $\mathbf{X}$, the standard error of $\beta_j$ can be computed as[13]

[13] Here $j = 0$ corresponds to the intercept $\beta_0$.

$$SE(\widehat{\beta}_j) = \sigma\sqrt{(j+1) - \text{th diagonal element of}(\mathbf{X}^T\mathbf{X})^{-1}}, j = 0, 1, \ldots, p.$$

A special case is the simple linear regression, where the standard errors, given fixed values of $x_1, \ldots, x_n$ are

$$SE(\widehat{\beta}_0) = \sqrt{\sigma^2\left[\frac{1}{n} + \frac{\bar{x}}{\sum_{i=1}^n (x_i - \bar{x})^2}\right]}, \; SE(\widehat{\beta}_1) = \sqrt{\frac{\sigma^2}{\sum_{i=1}^n (x_i - \bar{x})^2}}.$$

Examining the expression of standard errors of $\widehat{\beta}_0$ and $\widehat{\beta}_1$ shows that SE will be smallest if the denominator $\sum_{i=1}^n (x_i - \bar{x})^2$ is maximized. Thus, $SE$ is smallest if the predictor values, $X_i$'s, are more spread out from their center.

Note that the standard error expressions depend on the error variance $\sigma^2$, which is an unknown quantity. We can estimate $\sigma^2$, and $\sigma$, using the residual sum of squares.

**Residual standard error (RSE)**

We estimate

$$\widehat{\sigma}^2 = \frac{RSS}{n - (p+1)}, \; \widehat{\sigma} = \sqrt{\frac{RSS}{n - (p+1)}},$$

where $p + 1$ is the number of columns in the model matrix $\mathbf{X}$. The resulting estimator of $\sigma$ is known as *residual standard error (RSE)*.

We then plug-in $\widehat{\sigma}^2$ in place of $\sigma^2$ in the expressions of standard errors, to obtain *estimated standard errors*, $\widehat{SE}(\widehat{\beta}_j)$. For simplicity of presentation, we will still denote the estimated standard error by $SE(\widehat{\beta}_j)$.

The denominator $n-(p+1)$ in the expression of $\widehat{\sigma}^2$ merits some discussion. We can view this number as "sample size - number of $\beta$ parameters in the mean function". In simple linear regression we have two parameters in mean function, and thus we have the term $n-2$. In general, with $p$ predictors, the total number of parameters in the mean function is $p+1$ (since we need to include the intercept). Thus we use $n-(p+1)$ 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:[14]

[14] Here the `vcov()` function produces the matrix $\sigma^2(\mathbf{X}^T\mathbf{X})^{-1}$.The function `diag()` then extracts the diagonal elements of the matrix, and then we we take the square root by using the `sqrt()` function.

```
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 $\widehat{\sigma} = 6.216$. 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 $\widehat{\beta}_j$: a set/range of values which contain the "true" value of $\beta_j$ with high probability.[15] This can be investigated by a $t$-statistic based confidence interval.
- Perform hypothesis tests to determine whether the $j$-the predictor has any linear association with $Y$, $H_0 : \beta_j = 0$ vs. $H_1 : \beta_j \neq 0$. This can be investigated using a $t$-test.
- Perform hypothesis tests to determine whether the *any* of the predictors has any linear association with $Y$, $H_0 : \beta_1 = \ldots = \beta_p = 0$ vs. $H_1$ : at least on $\beta_j$ is non-zero. We can use $F$-test to answer this question.

We discuss each of the items below.

[15] Probability computed over repeated sampling.

### *Confidence interval*

Without going into mathematical details, we can obtain a confidence interval for $\beta_j$ using the standard errors.

**Confidence interval for $\beta_j$**

A $100(1-\alpha)\%$ confidence interval for $\beta_j$ is

$$[\widehat{\beta}_j \pm t_{1-\alpha/2,n-(p+1)} SE(\widehat{\beta}_j)],$$

where $t_{1-\alpha/2,n-(p+1)}$ denotes the $1-\alpha/2$ quantile of a $t_{n-(p+1)}$ distribution.

Again, the number $n-(p+1)$ is the same as we see in the estimator $\widehat{\sigma}^2$. This number is called the *degrees of freedom* of the $t$-distribution used above.

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
```

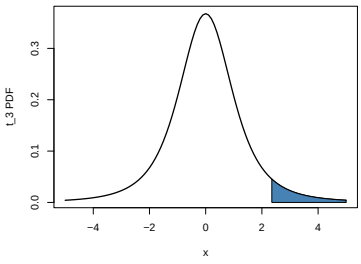



Figure 8: t PDF and quantiles. The shaded region has area p, and the x-axis value corresponding to the solid vertical line represents the (1-p)-quantile. In this example, we have p = 0.05, and the vertical line represents the upper 0.95-quantile.

```
##                 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 95% confidence. We can interpret the interval for the slope as follows: we estimate with 95% confidence that `medv` on average decreases between 1.03 and 0.87 for 1 unit increase in `lstat`.[16]

[16] Note that we are *not* stating that probability of the true value of $\beta_1$ falls between $-1.03$ and $-0.87$ is 95%. This is clearly a wring statement since the probability is either 0 or 1.

*t test*

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

$$H_0 : \beta_j = 0 \text{ vs. } H_0 : \beta_j \neq 0.$$

Note that $\beta_j = 0$ implies that $X_j$ is not in the model, and thus not associated with $Y$. We can use the $t$-statistic to perform the test:[17]

**$t$-test for $H_0 : \beta_j = 0$ vs. $H_0 : \beta_j \neq 0$**

The $t$ test statistic is

$$t = \frac{\widehat{\beta}_j - 0}{SE(\widehat{\beta}_j)}.$$

[17] In general, to test $H_0 : \beta_j = \delta$ vs. $H_0 : \beta_1 \neq \delta$, for any fixed value of $\delta$, we use the test statistic

$$t = \frac{\widehat{\beta}_j - \delta}{SE(\widehat{\beta}_j)}.$$

The test statistic measures how far away the estimated value of $\beta_1$ is from zero compared to the variability of the estimate measured by $SE(\widehat{\beta}_1)$. We expect the statistic $t$ to have a $t_{n-(p+1)}$ distribution *if* $H_0$ *is true*. Then we reject $H_0$ is the observed value of $t$ is very large or very small compared to what we expect from a $t_{n-(p+1)}$ distribution. Equivalently, we can compute the p-value of this test as

$$\text{p-value} = P(t_{n-(p+1)} > |t|) = 2[1 - F(|t|)],$$

where $F(|t|)$ is the CDF of the $t_{n-(p+1)}$ distribution evaluated at $|t|$. We reject $H_0$ if the p-value if smaller than $\alpha$, where we set $\alpha$ to be a small value (usually set to 5%). The quantity $\alpha$ is called the type I error of the test (probability of rejecting $H_0$ when it should not be rejected).

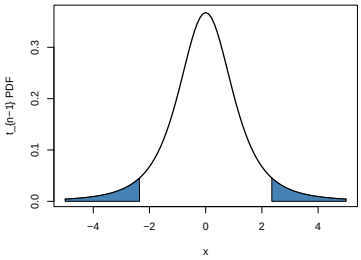



Figure 9: Two-tailed p-value for a t-test.

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 $\beta_1$ (coefficient of `lstat`) is very small, and thus reject $H_0$. 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 $X$ and $Y$ is most likely by chance even if $X$ and $Y$ are not actually related. A small p-value would indicate that it is unlikely to observe a large association between $X$ and $Y$ due to chance in absence of a real relationship.

Another way to test $H_0 : \beta_j = 0$ at $\alpha = 0.05$ is to check whether the value zero is in the 95% confidence interval of $\beta_j$ or not.[18] In the `Boston` data example with only `lstat` as predictor, zero is not in the 95% confidence interval of $\beta_1$, and thus the slope parameter is significantly different from zero with level $\alpha = 0.05$.

[18] In general, a level $\alpha$ test would correspond to a $100(1-\alpha)\%$ confidence interval.

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 $Y$ 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.

### *F test*

In the multiple linear regression with $p$ predictors, we investigate the whether the linear model is at all needed by testing $H_0 : \beta_1 = \ldots = \beta_p = 0$ vs. $H_1$ : at least on $\beta_j$ is non-zero. We can use $F$ test to do so. In general, we can test for *any subset* of the predictors using $F$-test, that is,

$$H_0 : \beta_{p-q+1} = \ldots = \beta_p = 0,$$

where we are testing the effects of the last $q$ predictors (last $q$ columns of $\mathbf{X}$).

**$F$-test for $H_0 : \beta_{p-q+1} = \ldots = \beta_p = 0$**

Let $RSS_0$ be the residual sum of squares of the model where we fit all the predictors *except the last q predictors*. Recall $RSS$ denotes the residual sum of squares for the full model. The $F$-statistic is

$$F = \frac{(RSS_0 - RSS)/q}{RSS/(n-(p+1))}.$$

We reject $H_0$ if the observed value of $F$ 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

$$Y_i = \beta_0 + X_{i1}\beta_1 + X_{i2}\beta_2 + X_{i1}X_{i2}\beta_3 + \epsilon,$$

where $X_{i1}$ and $X_{i2}$ 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, $H_0 : \beta_1 = \beta_2 = \beta_3 = 0$. 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 $H_0$, and the model is useful in predicting $Y$.

Alternatively, we can fit two models: the full model (`int_ols`) and another model with only intercept, and conduct $F$-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 $F$-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 $H_0 : \beta_2 = \beta_3 = 0$. 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 $F$-statistic 6.5425 with a p-value of 0.001567. If we use level $\alpha = 0.05$, we reject $H_0$, and can conclude that `age` does have association with response.

> When $q = 1$, that is, we are testing for one predictor only, the $t$-test and $F$-test are quivalent. In fact, the square of the $t$-statistic will give the $F$-statistic. Check this from summary of `simple_ols` when we test for only `lstat`.

So why do we need the $F$-test when we can examine $t$-test results for each predictor in our model? Can we not conclude the "model is useful in predicting response" if at least one $t$-test gives significant result without performing overall $F$-test? Indeed, we can not make such a claim. The main issue if *type I error* or *level* of the test. Typically, we set $\alpha = 0.05$ as level of the tests. That means, for individual $t$-tests, there is a 5% chance that some effect will be detected as significant when it is actually not significant. Thus, we might have about $0.05 * p$ many false significant results,[19] just by chance, even though none of the predictors might be useful. In fact, for large $p$, it is very likely we will observe at least one false significance just by chance, and incorrectly conclude that the model is useful. However,

[19] Especially when there are large number of predictors (large $p$).

the F-statistic does not suffer from this problem because it adjusts for the number of predictors. If $H_0$ is true, there is only a 5% 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, $R^2$.
- $F$ statistic discussed before

We have seen RSE as the estimator of $\sigma$ in the previous sections. In general, RSE quantifies the uncertainty in prediction on $Y$ from $X$ *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 $RSE = 6.22$. 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 6.22 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 ($R^2$) is another option to measure goodness of fit.

**Coefficient of determination: $R^2$**

Define the *total sum of squares (TSS)* as $\sum_i (Y_i - \bar{Y})^2$. Recall RSS is the residual sum of squares. Then

$$R^2 = 1 - \frac{RSS}{TSS}.$$

TSS measures the total variance in the response[20]. 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 $R^2$ as the *proportion of variance* in the response *explained by the model*. It can be shown that $0 \leq R^2 \leq 1$, with larger values indicting better fit. $R^2$ values close to zero would indicate that perhaps the linear model is wrong, and/or the error variance is high. Another way to interpret $R^2$ is that

$R^2 = (\text{correlation coefficient between observed and predicted values})^2$.

[20] Note that TSS is proportional to the sample variance of the $Y$'s.

### *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 10 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")
```

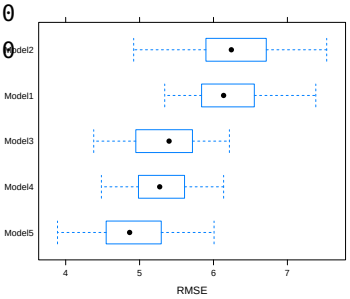



Figure 10: Boxplots of estimated test RMSE for different models.

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 $Y$ and $X$'s are indeed as posited by the linear regression model.
- Errors constant variance $\sigma^2$.
- 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 $X$ and $Y$ well. In the later case, transforming data (either $X$ or $Y$ or both) might prove useful. Alternatively, a non-linear/non-parametric regression might be considered as well.

In R, the `plot()` command will produce residual plot, along with other diagnostic plots for linear models.[21] Figure 11 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`$^2$ (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.

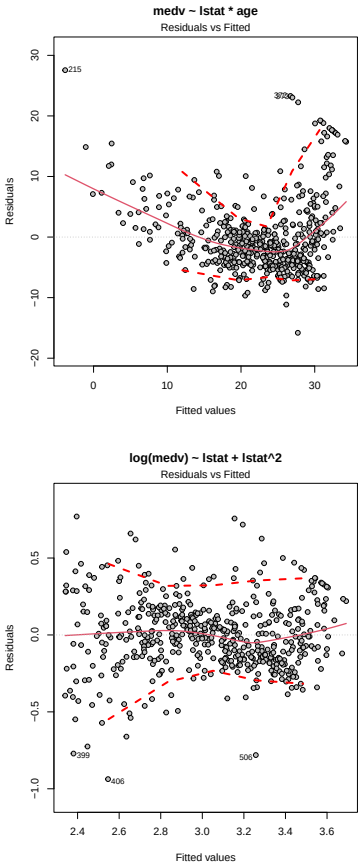



Figure 11: Residual vs Fitted values for two linear models fit on Boston data: regression of 'medv' on lstat, age and their two-way interaction (left plot), and regression of log(medv) on lstat and lstat$^2$ (right plot).

[21] See `?plot.lm()` for details.

## *Non-constant Variance of Errors*

The standard errors, confidence intervals, and hypothesis testing procedures discussed so far depends on the assumption of constant variance of the errors: $var(\epsilon_i) = \sigma^2$. We call such errors *homoscedastic*. If errors have different variance, such phenomenon is called *heteroscedasticity*. In the residual plot (bottom panel) in Figure 11, the black dashed lines track the 5% and 95% 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 12.

## *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).

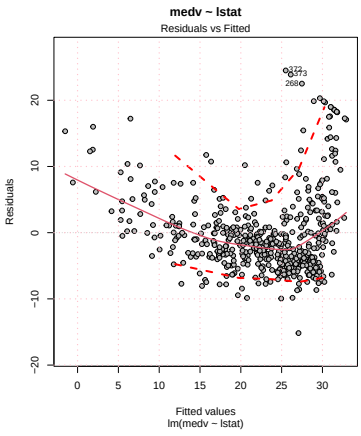



Figure 12: Example of a residual plot with non-constant error variance

### *Influential points*

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

Recall our linear model is $\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$. We estimate the regression parameter as $\widehat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$. Thus we we can predict the entire response vector $\mathbf{Y}$ by plugging-in $\widehat{\boldsymbol{\beta}}$ as

$$\widehat{\mathbf{Y}} = \mathbf{X}\widehat{\boldsymbol{\beta}} = [\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T]\mathbf{Y} = \mathbf{H}\mathbf{Y},$$

where $\mathbf{H} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$. The matrix $\mathbf{H}$ is called the *hat matrix*. Detection of influential points depend on the following two results:

> It can be shown that variance of the $i$-th residual, $var(Y_i - \widehat{Y}_i) = \sigma^2(1 - \mathbf{H}_{ii})$, where $\mathbf{H}_{ii}$ is the $i$-th diagonal entry of $\mathbf{H}$.
>
> The $i$-th diagonal entry of $\mathbf{H}$, $\mathbf{H}_{ii}$, is called the *leverage* of the $i$-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[22] are possible outliers. The function `rstudent()` can be used to compute studentized residuals. Figure 14 shows an example of such a plot. The point with absolute residual of more than 4 might be a potential outlier.

We can plot the studentized residuals vs. leverage statistics to detect possible high leverage points. Figure 15 shows an example of such a leverage plot. It can be shown that value of leverage leverage statistic is always between $1/n$ and 1. Also, the average value of leverages for all the observations is $(p+1)/n$, where $p+1$ is the number of columns in the model matrix $\mathbf{X}$. Thus, an observation can be a potential high leverage point if a given observation has a leverage statistic much larger that $(p+1)/n$. In Figure 15, we have $n = 506$ and $p = 2$, and thus $(p+1)/n = 0.0059289$. The point to the far right of the plot with leverage more that 0.10 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.

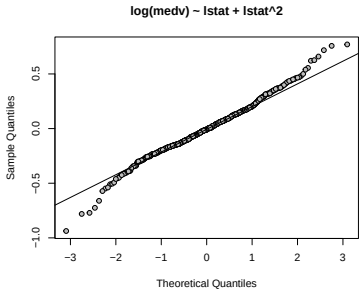



Figure 13: Normal Q-Q plot of residuals.

[22] A rule of thumb could be that possible outliers are observations with studentized residuals more than 3 in absolute value.

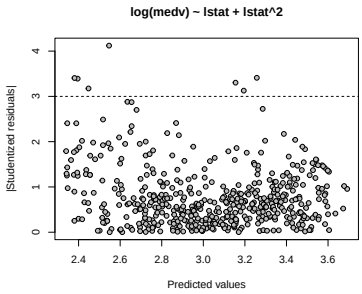



Figure 14: Example of a plot of absolute studentized residuals vs. fitted values.

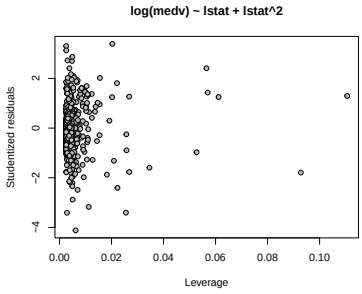



Figure 15: Example of a plot of absolute studentized residuals vs. leverage.

| 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 16 shows the correlation plot of Boston data, where we see that indeed `tax` and `rad` have high correlation.

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)*.

**Variance inflation factor (VIF)**

The variance inflation factor is the ratio of the variance of $\widehat{\beta}_j$ when fitting the full model to the variance if fit on its own. The VIF can be computed as follows:

$$VIF = (1 - R_j^2)^{-1},$$

where $R_j^2$ is the $R^2$ value from a regression of $X_j$ onto the remaining predictors.

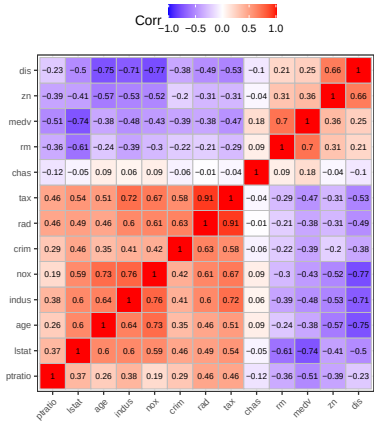



Figure 16: Correlation plot of Boston data.

The minimum value of VIF is 1. 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 $x_1, \ldots, x_p$ as

$$\widehat{Y} = \widehat{\beta}_0 + x_1\widehat{\beta}_1 + \ldots + x_p\widehat{\beta}_p.$$

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 $lstat = 5$ for the simple linear regression of `medv` on `lstat`: `fit` is the point prediction, `lwr` and `upr` are the lower and upper bound of the prediction interval, respectively. Note the function argument `interval = "prediction"`.[23]

[23] See `?predict.lm` for details.

```
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 $E(Y|x_1, \ldots, x_p)$. However, the prediction interval is not the same as the confidence interval of $E(Y|x_1, \ldots, x_p)$. This is because predicting the actual response $Y$ is more difficult that estimating the mean $E(Y|X)$. For the ideal case where we know the exact values of $\beta_0, \ldots, \beta_p$, then $E(Y|x_1, \ldots, x_p)$ is exactly determined. But even then, the response $Y$ has some variability due to error $\epsilon$. 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 $E(Y|x_1, \ldots, x_p)$ 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 95% 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 $X$'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

$$Y_i = \beta_0 + X_{i1}\beta_1 + X_{i2}\beta_2 + \epsilon_i,$$

where $Y_i$ denotes `medv`, $X_{i1}$ denotes `lstat` and $X_{2i}$ denotes `chas`. This model effectively represents two lines, one for each value of `chas`. The display below gives the two regression lines:

$$chas = 0 : E(Y_i|X_{1i}, X_{2i} = 0) = \beta_0 + X_{i1}\beta_1.$$

$$chas = 1 : E(Y_i|X_{1i}, X_{2i} = 1) = (\beta_0 + \beta_2) + X_{i1}\beta_1.$$

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 17. The solid red line corresponds to `chas = 0` and the dashed black line for `chas = 1`.

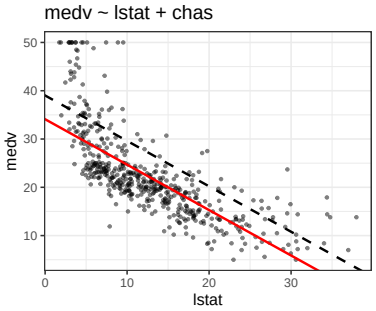



Figure 17: Regression line for chas = 0 (solid red) and 1 (dashed black) based on a linear model with main effects of chas and lstat.

If we include an interaction term:

$$Y_i = \beta_0 + X_{i1}\beta_1 + X_{i2}\beta_2 + X_{i1}X_{i2}\beta_3 + \epsilon_i,$$

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

$$chas = 0 : E(Y_i|X_{1i}, X_{2i} = 0) = \beta_0 + X_{i1}\beta_1.$$

$$chas = 1 : E(Y_i|X_{1i}, X_{2i} = 1) = (\beta_0 + \beta_2) + X_{i1}(\beta_1 + \beta_3).$$

See Figure 18 for the fitted lines.

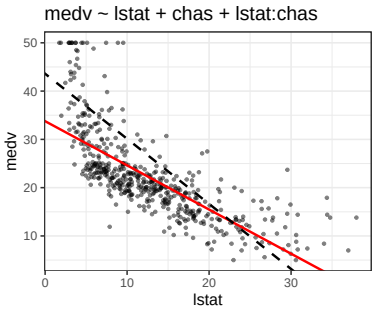



Figure 18: Regression line for chas = 0 (solid red) and 1 (dashed black) based on a linear model with main effects of chas and lstat, as well as their interaction.

The ideas presented above can be generalized to categorical variables with more that two levels. Suppose that $Z_i$ is a variables with three levels, "L1", "L2" and "L3". Then we need to create $3 - 1 = 2$ *dummy variables*:[24]

$$Z_{i1} = I(Z_i = "L1"), \text{ and } Z_{i2} = I(Z_i = "L2")$$

We do not need a new dummy for level "L3" since $Z_{i1} = Z_{i2} = 0$ 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

[24] Recall that for any event $A$, we define the *indicator function* $I(A) = 1$ if $A$ is true, 0 otherwise.

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.[25] 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 309 predictors, not simply 82. If the sample size were smaller, say $n = 500$, standard techniques like 5-fold CV (training set size will be 400) or $70\% - 30\%$ split (training set size 350) 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.

[25] See `?ames_raw` after loading `AmesHousing` package.

## *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 $R^2$ from the training set in model selection is undesirable as they will always choose the largest model possible – minimum RSE and maximum $R^2$ 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 $p = 12$. If we want to investigate performance of all possible subsets, we have to go though $2^{12} = 4096$ models. On top of that if we want to use 5-fold CV, repeated 50 times, we have to fit a total of $2^{12} \times 5 \times 50 = 1.024 \times 10^6$ models.

Alternatively, there are metrics available that adjusts training performance metrics such as $RSS$ and $R^2$ 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 $R^2$,
- Akaike information criterion (AIC),
- Bayesian information criterion (BIC), and
- $C_p$ statistic.

Adjusted $R^2$ 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 $C_p$ *adds a penalty term* involving number of predictors to the training RSS to account for model size.

Suppose we have a model with $d$ predictors. Recall that $R^2 = 1 - RSS/TSS$. Adjusted $R^2$ is defined as

$$\text{Adjusted } R^2 = 1 - \frac{RSS/(n-d-1)}{TSS/(n-1)},$$

where $d$ is the number of predictors in the model. Maximizing the adjusted $R^2$ is equivalent to minimizing $RSS/(n-d-1)$. Unlike $RSS$, which monotonically decreases as $d$ increases, $RSS/(n-d-1)$ will increase and decrease as $d$ changes. We choose the model with maximum adjusted $R^2$.

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

$$[RSS + P(n, d, \widehat{\sigma}^2)]/n,$$

where $P(n, d, \widehat{\sigma}^2)$ 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 $P$:[26]

$$P(n, d, \widehat{\sigma}^2) = \begin{cases} 2d\widehat{\sigma}^2, & \text{for } C_p, AIC \\ log(n)\, d\, \widehat{\sigma}^2, & \text{for } BIC \end{cases}$$

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

[26] See Introduction to Statistical Learning, Chapter 6.1.3.

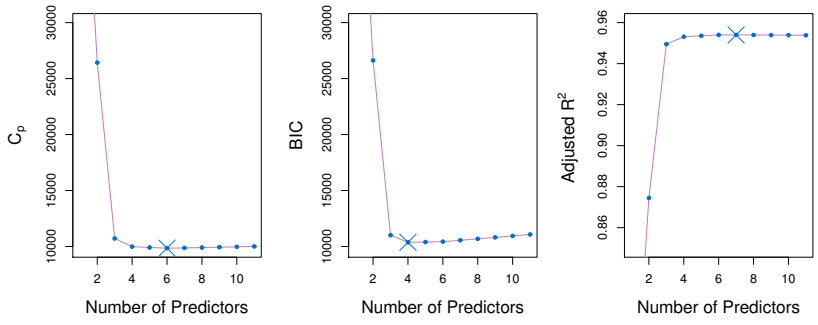



Figure 19: Example of model selection using AIC/$C_p$, BIC and adjusted $R^2$.

It seems AIC and $C_p$ 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/$C_p$ as $n$ increases. Thus BIC tends to produce smaller models compared to AIC/$C_p$. Figure 19 shows an example of model selection using AIC/$C_p$, BIC and adjusted $R^2$.

### *Best subset selection*

In this approach, we need to fit a separate least square model to *each* of the possible combination of the $p$ predictors in the dataset, that is, we need to fit all $2^p$ 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 $M_0$.
2. For $k = 1, \ldots, p$, fit all $C_k^p$ models with $k$ predictors, and pick the best model (smallest RSE, largest $R^2$ etc.). Denote the resulting model as $M_k$.
3. Among the models $M_0, M_1, \ldots, M_p$, choose the best model using AIC, BIC, adjusted $R^2$ or CV.

Note that we can use cross-validation for the entire set of $2^p$ possible models if we have such computational resources (for larger $p$, 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 $2^p$ possible models to $p + 1$ 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 $R^2$ 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 $R^2$ occurs for model size 10. 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 $2^p$ models can be computationally intensive for large values of $p$. The following two approaches provide computationally efficient alternatives using *stepwise subset selection*.

### *Forward Stepwise Selection*

Recall that the best subset selection procedure considers all $2^p$ possible models containing subsets of the $p$ predictors. In contrast, *forward stepwise selection* considers a much smaller set of models. The algorithm as as follows:

1. Start with the model with only intercept, and no other predictor. Denote the model by $M_0$.
2. For $k = 0, \ldots, p-1$,
   - consider all $p-k$ models that adds one more predictor to the existing predictors in $M_k$.
   - choose the best among these $p-k$ models; denote this model by $M_{k+1}$
3. Among the models $M_0, M_1, \ldots, M_p$, choose the best model using AIC, BIC, adjusted $R^2$ or CV.

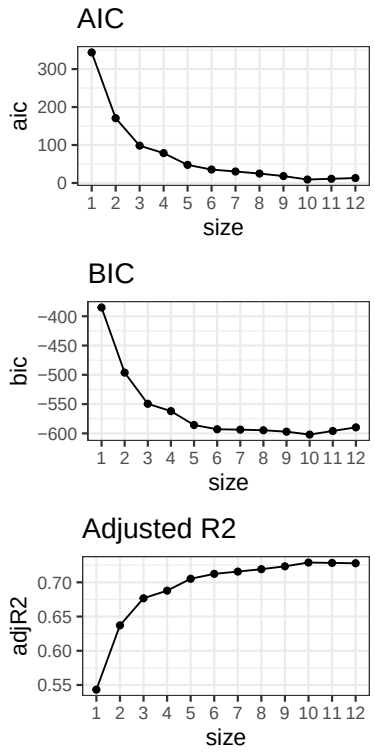



Figure 20: AIC, BIC and Adjusted $R^2$ for best subset selection in Boston data.

Forward stepwise selection involves fitting one intercept-only model, along with $p-k$ models in the $k$th iteration, for $k = 0, \ldots, p-1$. This reduces the computational complexity substantially from the best subset selection, which fits $C_k^p$ models for each $k = 1, \ldots, p$. 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
```

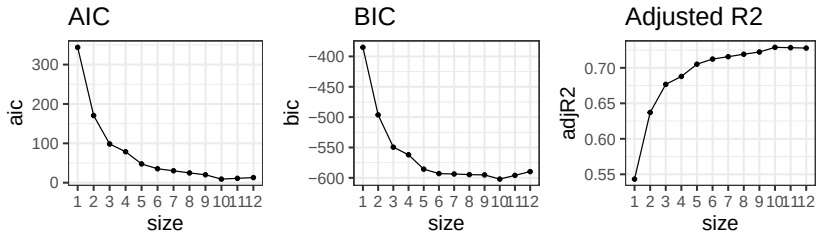



Figure 21: AIC, BIC and Adjusted $R^2$ for forward stepwise selection in Boston data.

*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 $M_p$.
2. For $k = p, p-1 \ldots, 1$,
   - consider all $k$ models that contain all but one of the predictors in $M_k$, for a total of $k-1$ predictors.
   - choose the best among these $k$ models; denote this model by $M_{k-1}$
3. Among the models $M_0, M_1, \ldots, M_p$, choose the best model using AIC, BIC, adjusted $R^2$ or CV.

Like forward stepwise selection, backward stepwise selection is not guaranteed to yield the best model containing a subset of the $p$ 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
```

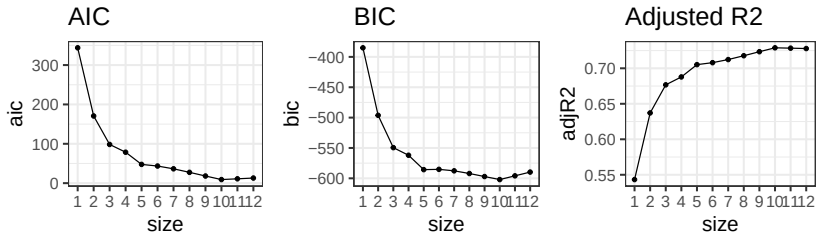



Figure 22: AIC, BIC and Adjusted $R^2$ for backward stepwise selection in Boston data.

In the `Boston` data example seen so far, results from using AIC, BIC and adjusted $R^2$ 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 $R^2$, 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 $2^p$ 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 $M_0, \ldots, M_p$ 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.

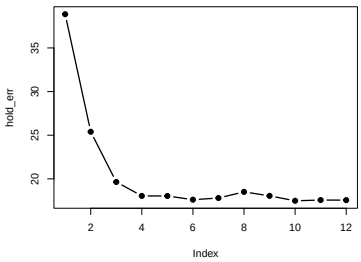
```
set.seed(1001)
## Create test and training sets
library(rsample)
data_split <- initial_split(Boston, prop = 0.8)
test_set <- testing(data_split)
train_set <- training(data_split)

## 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 $V$-fold cross-validation as follows:

- Split the data into $V$ equally sized folds.
- For $v = 1, \ldots, V$:

  - Set $v$-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 $V$ 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.

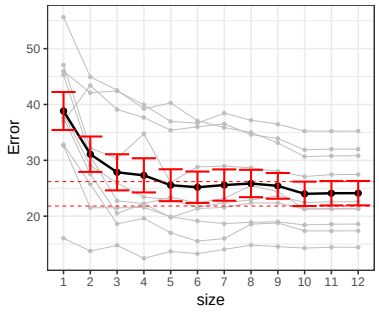



Figure 23: Best subset selection using 5-fold cross-validation. The gray lines are test MSE profiles for the 5 folds. The black line is the mean test MSE over the folds. The red error bars indicate test MSE +/- one SE.

Figure 23 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 23, 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 $p$ 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

$$\sum_i (Y_i - \beta_0 - X_{i1}\beta_1 - \dots X_{ip}\beta_p)^2 + P,$$

where the term $P$ 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[27]

$$\sum_i (Y_i - \beta_0 - X_{i1}\beta_1 - \ldots - X_{ip}\beta_p)^2 + \lambda \sum_{j=1}^{p} \beta_j^2,$$

where $\lambda \geq 0$ is a tuning parameter. The penalty term $\lambda \sum_{j=1}^{p} \beta_j^2$ is called a *shrinkage penalty*.[28] Here $\lambda$ controls the relative impact of the two terms on the regression coefficient estimates. For large values of $\lambda$, the quadratic penalty term dominates the criterion, and the resulting estimates approach to zero. When $\lambda = 0$, there is no penalty, and thus we get exactly the ordinary least squares estimates. Thus we must select a reasonable value of $\lambda$ to balance both the terms.

Recall that $\mathbf{X}$ 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):[29]

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

Notice again that setting $\lambda = 0$ gives us the least squares estimates, $\widehat{\beta}$. Also note that, for $\lambda > 0$, the matrix $(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})$ always has an inverse *even if* $\mathbf{X}$ *does not have full column rank*. Thus, even in presence of collinearity/redundant columns in $\mathbf{X}$, ridge regression will still produce unique regression estimates.[30]

Figure 24 shows the estimated ridge regression coefficients for different values of $log(\lambda)$ in `Boston` data with `medv` as response, and *standardized* predictors. The left most part of the plot corresponds to $\lambda = 0$, and shows the least squares estimates. The right extreme of the plot represents a large value of $\lambda$, and we see that all the coefficients are very close to zero.

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

$$\text{minimize} \sum_i (Y_i - \beta_0 - X_{i1}\beta_1 - \ldots - X_{ip}\beta_p)^2$$

subject to the constraint

$$\sum_{j=1}^{p} \beta_j^2 \leq t,$$

[27] Note that the intercept $\beta_0$ is not penalized.

[28] The idea of using the sum-of-squares of the parameters as penalty is also used in neural networks – it is known as *weight decay*.

[29] Here $\mathbf{I}$ denotes the identity matrix: a diagonal matrix with all diagonal elements being 1.

[30] 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.

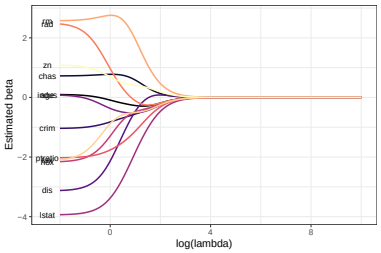



Figure 24: Ridge regression coefficients for different values of lamda (log10 scale) for Boston data.

for some $t > 0$. The second formulation of ridge regression explicitely puts constraint on the size of the regression coefficients. The parameters $\lambda$ in the penalized formulation and $t$ 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 $t$ determines how large/small regression coefficients can become. If $t$ is set to very large, then we are effectively allowing $\beta$'s to take any value (equivalent to setting a small $\lambda$). On the other hand, a small $t$ will force the $\beta$'s to be smaller and closer to zero (equivalent to setting large $\lambda$).

In presence of multicollinearity, the corresponding $\beta$'s can become wildly variable. A very large positive $\beta$ on one variable can be canceled by a similarly large negative $\beta$ on another predictor correlated to the first one. A size constraint imposed by $t$, 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., $R^2$, $MSE$ etc). If we multiply a predictor by a constant $c$, then the resulting least square coefficient estimate will get multiplied by $1/c$. In other words, using least squares, the quantity $X_j\widehat{\beta}_j$ will remain the same no matter how we scale the $j$-th predictor.[31]

[31] This is the reason we call least squares estimators *scale equivariant*.

```
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 $\widehat{\beta}_j^{\text{ridge}}$ will depend on the scaling of the $j$-th predictor, the value of the tuning parameter $\lambda$, *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 $\beta_0$. 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 $c$ to each of the responses $Y_i$, the resulting predictions also shift by the same amount $c$. But this does not happen in ridge regression if we penalize the intercept – therefore we do not penalize $\beta_0$.

It can be shown that, if we center each covariate, that is, we use $X_{ij} - \bar{X}_j$ as predictors, then the estimator of the intercept is simply the sample mean of $Y$: $\widehat{\beta}_0 = \bar{Y}$. The remaining coefficients, $\beta_1, \ldots, \beta_p$, are estimated by a ridge regression without intercept.

> For simplicity, we will henceforth assume that the model matrix $\mathbf{X}$ does not include intercept, and thus has only $p$ columns, not $p+1$. We will also assume that mean of each column is zero.

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

In R, we can use the `glmnet()` function in the `glmnet` library.[32] 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).

[32] See `?glmnet` for more details.

```
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 $\lambda$ values.[33] For each value of $\lambda$, the output `betahat` contains the corresponding estimates of the regression coefficients. Figure 24 shows the estimated coefficients for different values of $\lambda$.

[33] `glmnet()` has a default way to set $\lambda$ values as well if we do not specify $\lambda$ manually.

How do we choose the "optimal" value of $\lambda$? We again come back to *bias-variance trade-off*. Note that the penalty parameter $\lambda$ effectively controls the model complexity: small values of $\lambda$ results in close to least squares fit (lower bias, higher variance), while large values of $\lambda$ results in almost an intercept-only model (higher bias, lower variance). Figure 25 shows bias-variance trade-off of ridge regression.

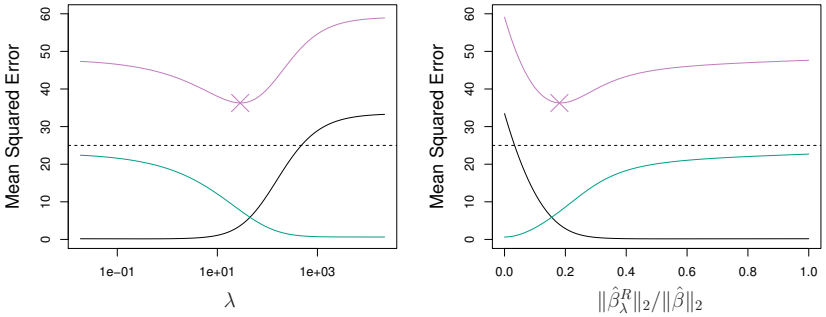



Figure 25: Bias-variance trade-off of ridge regression. Figure taken from extitIntroduction to Statistical Learning. Displayed are squared bias (black), variance (green), and test mean squared error (purple) for the ridge regression predictions on a simulated data set. The horizontal dashed lines indicate the minimum possible MSE.

Ideally, we would like to select $\lambda$ 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 $\lambda$, and compute the cross-validation (or holdout) error for each value. The optimal $\lambda$ is the one with minimum test error. Finally, we refit the model to the full data using the optimal $\lambda$.

We can use `glmnet.cv()` function to perform cross-validation. By default, `glmnet.cv()` uses 10-fold CV.[34]

[34] See `?glmne.cv()` for details. We can use `nfolds` argument to specify number of folds while using CV.

```
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 26 shows the results.

```
# Plot cv results
plot(cv_out)
```

The "best" value of $\lambda$ can chosen by minimizing the CV error. The left vertical line in Figure 26 represents this value. From Figure 26, we see that there are a range of $\lambda$ 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 $\lambda$: rather

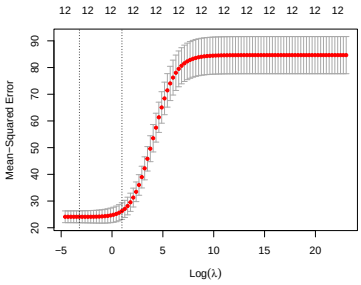



Figure 26: Cross-validation results for Boston data using ridge regression.

than choosing the $\lambda$ that gives the minimum test MSE, we would pick the largest $\lambda$ (less model complexity) whose test MSE is within one standard error of the minimum test MSE. The right vertical line in Figure 26 represents this value. The two values $\lambda$ 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
```

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 $p$ is close to $n$. When $p > n$, 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 $\lambda$. 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.

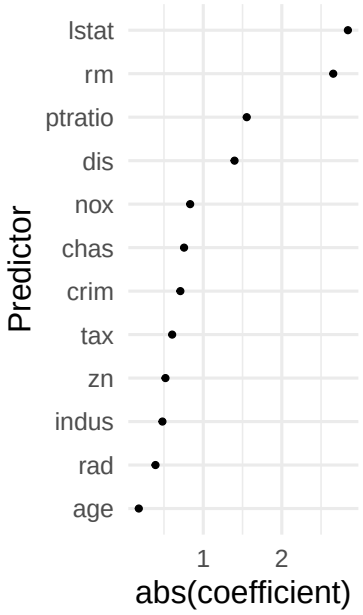



Figure 27: Predictors arranged by absolute values of their estimated coefficients using 1-SE rule.

*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 $\beta_j$ are obtained by minimizing

$$\sum_i (Y_i - \beta_0 - X_{i1}\beta_1 - \ldots - X_{ip}\beta_p)^2 + \lambda \sum_{j=1}^{p} |\beta_j|,$$

for $\lambda \geq 0$. Due to the $L_1$ penalty term, there is no closed form solution to the lasso problem.[35] An equivalent way to write the LASSO problem is in the form of a constrained minimization problem,

$$\text{minimize } \sum_i (Y_i - \beta_0 - X_{i1}\beta_1 - \ldots - X_{ip}\beta_p)^2$$

subject to the constraint

$$\sum_{j=1}^{p} |\beta_j| \leq t,$$

for some $t > 0$.

Much like ridge regression, lasso also shrinks the regression coefficients towards zero. However, due to the $L_1$ penalty term $\sum_j |\beta_j|$, some of the coefficients will be shrunk exactly to zero. It is easier

[35] Computing the lasso solution is a *quadratic programming problem*. Efficient algorithms are available for computing the entire path of solutions as $\lambda$ is varied. These algorithms have the same computational cost as for ridge regression. Interested readers should see *Elements of Statistical Learning* for details.

to see if we have standardized the predictors, and if they are orthogonal to each other. In that case, the explicit lasso solution is $\widehat{\beta}_j^{\text{lasso}} = sign(\widehat{\beta}_j)(|\widehat{\beta}_j| - \lambda)_+$. 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 $t$, we will effectively get the least squares estimates. Specifically, it can be shown that if $t$ is chosen larger that $t_0 = \sum_{j=1}^{p} |\widehat{\beta}_j|$, then lasso estimates are identical to least squares estimates. On the other hand, if we chose $t = t_0/2$, then the least squares estimates are shrunk, on average, by about 50%. Figure 28 shows the reason some lasso estimates are exactly set to zero while ridge estimates are not. Here $\widehat{\beta}$ represents least squares solution while while the blue diamond and circle represent the lasso and ridge regression constraints. For large values of $t$, the constraint region will contain $\widehat{\beta}$ and thus both ridge and lasso estimates will be identical to least squares (equivalently choosing $\lambda = 0$). For smaller values of $t$, the least squares estimate may lie outside the constraint region, like we see in Figure 28.

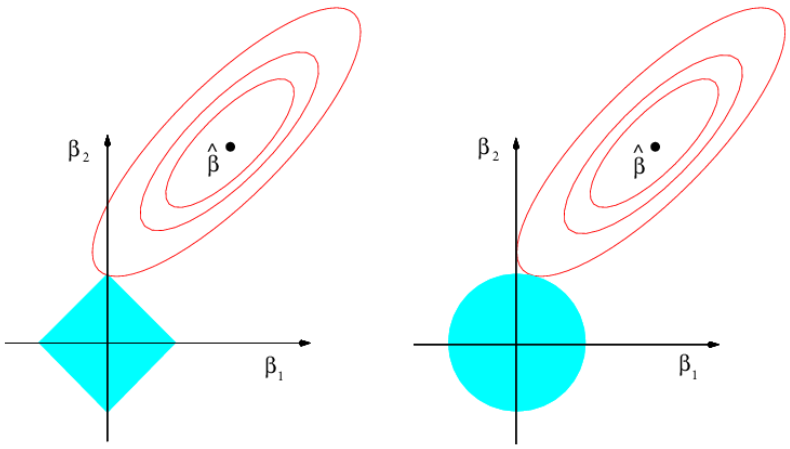



Figure 28: Contours of the error and constraint functions for the lasso (left) and ridge regression (right). The solid blue areas are the constraint regions for lasso and ridge, while the red ellipses are the contours of the RSS. Figure taken from *Introduction to Statistical Learning*.

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 28, we have $\beta_1 = 0$.

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 29 shows the estimated regression coefficients as $\lambda$ changes. The left extreme of the plot corresponds to least squares fit ($\lambda = 0$).

```
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
```

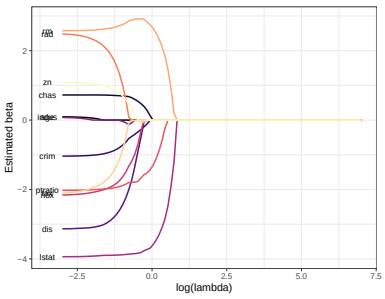



Figure 29: Lasso regression coefficients for different values of lambda (log10 scale) for Boston data.

Like ridge regression, we need to carefully select $\lambda$. 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)
```

Figure 30 shows the results of selection of $\lambda$ using 10-fold cross-validation. The $\lambda$ values with minimum CV error and chosen by the one standard rule are shown below, along with the corresponding coefficient estimates.

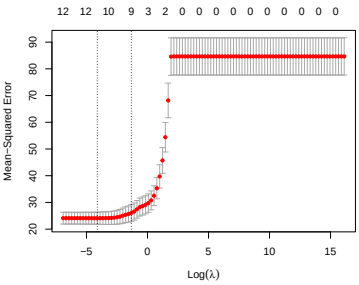



Figure 30: Cross-validation results for Boston data using lasso regression.

```
## 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
```

```
## ## 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
```

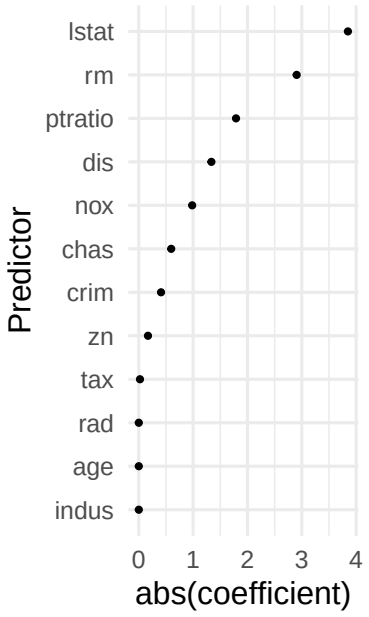



Figure 31: Predictors arranged by absolute values of their estimated coefficients using 1-SE rule from a lasso fit.

Notice that the coefficient of `indus` is exactly set to zero, and is thus excluded from the final model, when we choose $\lambda$ by minimizing CV

error. The one standard error rule gives a much larger $\lambda$, 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*,[36] which minimizes

$$\sum_i (Y_i - \beta_0 - X_{i1}\beta_1 - \ldots - X_{ip}\beta_p)^2 + \lambda((1-\alpha)\sum_{j=1}^{p}\beta_j^2 + \alpha\sum_{j=1}^{p}|\beta_j|),$$

for $\lambda \geq 0$ and $\alpha \in [0,1]$. Note that lasso and ridge regressions are special cases of elastic net for $\alpha = 1$ and $\alpha = 0$, respectively.[37] 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.[38] 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 $p > n$, and predictors (genes) are often have high correlation among them.

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

[36] 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.

[37] This is the formulation `glmnet()` uses with the `alpha` argument.

[38] See *Elements of Statistical Learning* for more discussion.

### *Other variable selection methods*

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

- *adaptive lasso*:[39] 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*:[40] 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*:[41] 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

[39] Zou, H (2012). The Adaptive Lasso and Its Oracle Properties, JASA, 101, 1418 - 1429

[40] 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

[41] 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.

is a function of time (functional data or time series). We can use the `genlasso` package here.

- *Smoothly clipped absolute deviations (SCAD)*[42] 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.

[42] Fan J and Li R. (2001). Variable Selection via Nonconcave Penalized Likelihood and its Oracle Properties. Journal of American Statistical Association, 96:1348 - 1360.

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 $X_{i1}, \ldots, X_{ip}$. A typical dimension reduction method has two steps:

1. Create new predictors $Z_{i1}, \ldots, Z_{iM}$ by transforming/combining the original predictors. Usually we choose $M < p$, and thus reducing the dimension of the problem.
2. Fit the regression model with the new $M$ predictors:

$$Y_i = \theta_0 + Z_{i1}\theta_1 + \ldots + Z_{iM}\theta_M + \epsilon_i.$$

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.

*Overview of PCA*

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 $p$ predictors $\mathbf{X}_i = (X_{i1}, \ldots X_{ip})^T$. The main goal of PCA is to identify *linear combinations* of the form

$$Z_{im} = a_{m1}X_{i1} + \ldots + a_{mp}X_{ip}, m = 1, 2, \ldots, M,$$

that explain most of the variability in the data.[43] Typically we choose $M < p$, and the new variables, $Z_{im}$, are ordered according to their importance. Specifically, $Z_{i1}$ is designed to capture the most variability in the original variables by any linear combination – this is called the *first principal component (PC)*. Then $Z_{i2}$, the *second PC*, captures the most of the *remaining* variability while being *uncorrelated* to $Z_{i1}$. We continue until we have the $p$-th PC $Z_{ip}$. In the end, we hope that the first few PCs, $Z_{i1}, \ldots, Z_{iM}$, will capture most of the variability in the original predictors.

[43] Mathematically, we need to *normalize* the weights, that is, we ensure that $a_{m1}^2 + \ldots + a_{mp}^2 = 1$.

**Principal component directions and Loadings**

The vectors $\boldsymbol{a}_m = (a_{m1}, \ldots, a_{mp})^T$ are called the *principal component (PC) directions*. The individual components (the weights $a_{m1}, \ldots, a_{mp}$) of each PC direction are called *loadings*. The loadings tell us how the original variables are weighted to get the new variables.

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 $2.8404759 \times 10^4$ 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 100, it does not make `lstat` any more important than it originally was, but its variance will be inflated by a factor of $10,000$ making `lstat` dominant over the rest of the predictors. To avoid this issue, we will standardize each predictor. [44] Since now every predictor will have variance one, total variation is simply the number of predictor in the data.

[44] This is my general recommendation when performing PCA.

```
# 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 $\mathbf{X}$). 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 $Z$ contain one PC – the first column if for PC1, the second for PC2 and so on. First we note that the combined variation in $Z$ 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.

$$\text{Proportion of TV captured by } j\text{-th PC} = \frac{var(Z_j)}{TV}.$$

```
# 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.

$$\text{Proportion of TV captured by first } j \text{ PCs} = \frac{var(Z_1) + \ldots + var(Z_j)}{TV}.$$

```
# 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 100% of $TV$, 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 15% of TV (i.e., capture 85% 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.[45]

[45] We will discuss more about interpreting the loadings in a later chapter.

```
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 32) 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.

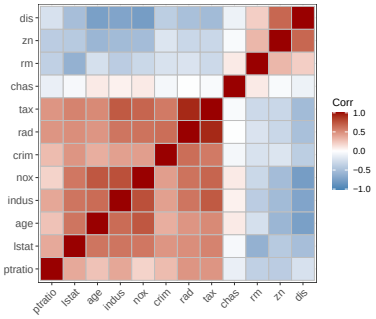



Figure 32: Correlation plot of Boston data.

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 $p = 2$. Thus for PC1 we need to determine loading $a_{11}, a_{12}$ so that variance of $a_{11}X_1 + a_{12}X_2$ is maximized. The condition on the loadings is

$$a_{11}^2 + a_{12}^2 = 1.$$

This is the equation of a circle, centered at zero, with radius one. So we only need to look at points $(a_{11}, a_{12})^T$ that are on the perimeter of the circle. This is what we mean by *direction*; see Figure 33.

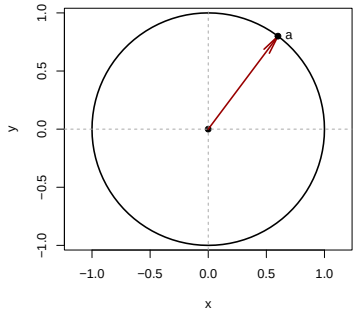



Figure 33: First PC direction.

Thus, given a data scatterplot, the 1st PC points to the direction along with most of the variation lies. In Figure 34, 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).

Let us now consider the case with three variables, $p = 3$. In this case, the loadings are $a_{11}, a_{12}, a_{13}$ and the constraint becomes

$$a_{11}^2 + a_{12}^2 + a_{13}^2 = 1.$$

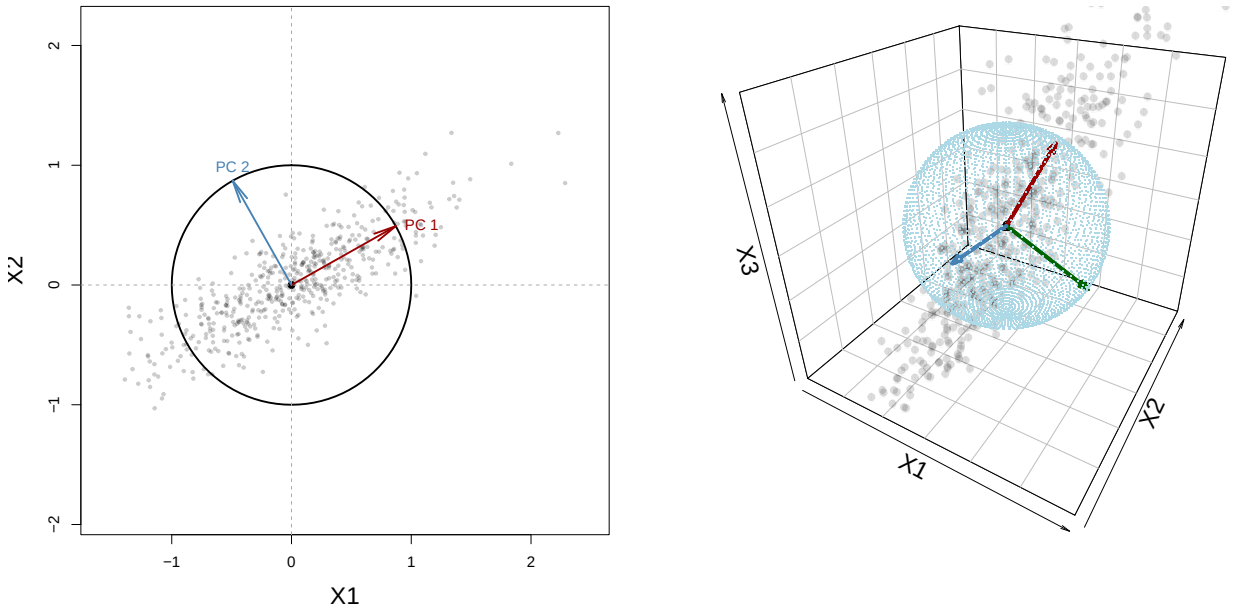



Figure 34: Geometry of PCA in two and three dimensions (left and right panels, respectively).

This is the equation of a sphere, centered at zero, with radius one. Thus we only need to look at points $(a_{11}, a_{12}, a_{13})^T$ that are on the *surface of the sphere*.

Now consider a data scatter in three dimensions (gray points in Figure 34, 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 $\boldsymbol{a}$ is also represented by $-\boldsymbol{a}$ (just like "x-axis" corresponds to both positive and negative directions). Thus if $\boldsymbol{a}$ is a PC then so is $-\boldsymbol{a}$. In other words, if $Z_1 = a_{11}X_1 + \ldots + a_{1p}X_p$ is a PC, then so is $Z_1' = -a_{11}X_1 - \ldots - a_{1p}X_p$. 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`.

*Performing Principal Components Regression (PCR)*

Now that we have constructed the PCs, we can choose the first $M$ PCs, $Z_{i1}, \ldots, Z_{iM}$, and build a regression model with the PCs as

predictors. Here we are assuming that the direction that the original predictors, $X_1, \ldots, X_p$ show most variation are in fact the directions associated with the response.[46]

[46] There is no assurance such an assumption actually holds.

If the assumption above holds true, then using PCR with $Z_{i1}, \ldots, Z_{iM}$ as predictors will give a better result than using all the $p$ original predictors. PCR may also help mitigating overfitting.

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

In R, we can use the `pcr()` function in the `pls` package.[47] Note the usage of arguments `center = TRUE` and `scale = TRUE`.

[47] Other packages such as `caret` can also so this.

```
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 $M$.

```
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.

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`.

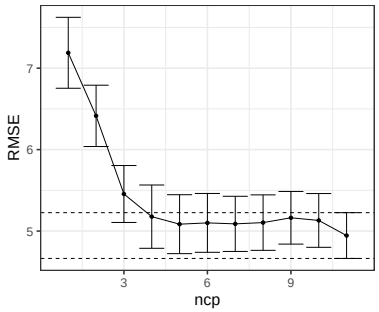



Figure 35: Cross-validation error with one SE error bars.

```
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.[48]

[48] Recall, for a random sample $X_1, \ldots, X_n$, standard error of sample mean is

$$SE(\bar{X}) = \frac{\text{sample SD of X values}}{\sqrt{\text{sample size}}}.$$

```
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 35.

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

$$Y_i = \theta_0 + \sum_{m=1}^{4} Z_{im}\theta_m + \epsilon_i$$

Since $Z_{im} = a_{m1}X_{i1} + \ldots + a_{mp}X_{ip}$, we can write the model above in terms of the original variables as

$$Y_i = \theta_0 + [\sum_{m=1}^{4} a_{m1}\theta_m]X_{i1} + \ldots + [\sum_{m=1}^{4} a_{mp}\theta_m]X_{ip} + \epsilon_i.$$

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, $X_1, \ldots, X_p$ 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)[49] 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.

[49] 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`.

Recall, we are still operating under the assumption that the predictors have been standardized. PLS begins by performing a *simple linear regression* of $Y_i$ on the $j$-th original predictor, $X_{ij}$, for each $j = 1, \ldots, p$. The resulting estimates of slopes are denoted as $a_{11}, \ldots, a_{1p}$, respectively. Then the first PLS component is constructed as

$$Z_{i1} = a_{11}X_{i1} + \ldots + a_{1p}X_{ip}.$$

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: $Z_{i2} = a_{21}X_{i1} + \ldots + a_{2p}X_{ip}$, where $a_{2j}$ if estimated regression coefficient of $X_{ij}$ from the simple linear regression of the residuals (obtained above) on $X_{ij}$. We continue this process until we have all the $p$ PLS components. As in PCR, we take the leading $M$ PLS components. A multiple linear regression is then fitted with $Y$ as response and the $M$ PLS components as predictors.

We can use the `plsr()` function in `pls` package, or use `caret` with `mehod = "pls"`. The number of PLS components, $M$ 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 va
## The final value used for the model was ncomp = 8.
```

We can plot the estimated test errors and error bars representing plus/minus one standard error as we did in PCR – see Figure 36. Using minimum test error, we can use 8 PLS components. Using one standard error rule, it seems two PLS components are sufficient.

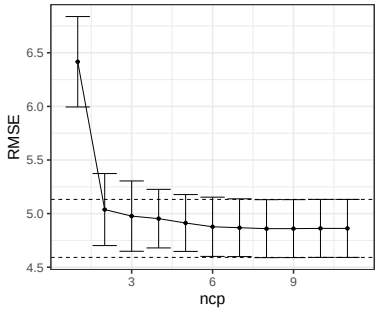



Figure 36: Cross-validation error with one SE error bars.

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, $Z_{im}$ 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.[50] [51] 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.

[50] 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.

[51] Frank, I.E. and Friedman, J.H. (1993) An Statistical View of Some Chemometrics Regression Tools. Technometrics, 35, 109 - 135.

## *High-dimensional data*

So far, all the methods we discussed assume that the number of predictors ($p$) is (much) less than the sample size ($n$). The performance of these methods deteriorate as $p$ gets closer or exceed $n$. Data sets containing more features than observations (or sometimes number of features slightly smaller than $n$) 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)*[52] 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.

[52] These are individual DNA mutations that are relatively common in the population

When we have $p > n$, usual least squares regression should not be performed. This is because as $p > n$, 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 $R^2$ and $RSE$ 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 $p$ predictors. When $p + 1 \geq n$ (or $p \geq n$ if intercept is not in the model), least squares gives a perfect fit with zero residuals ($R^2 = 1$ and $RSE = 0$). However, such a model will perform extremely poorly in a test set due to very high model variance. Figure 37 further illustrates the risk of carelessly applying least squares when the number of features is large.

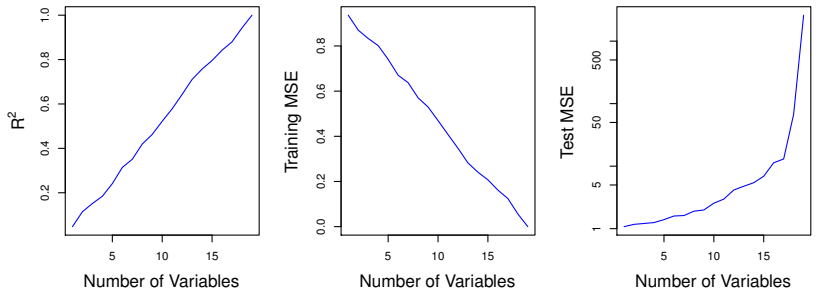



Figure 37: Risk of carelessly applying least squares when the number of features is high. Data were simulated with n = 20 observations, and regression was performed with between 1 and 20 features, each of which was completely unrelated to the response.

In fact, the model evaluation approaches that do not require a test set (AIC, BIC, adjusted $R^2$), are also not appropriate for in the high-dimensional setting due to instability of estimation of $\widehat{\sigma}^2$ and RSS, both of which will be zero when $p + 1 \geq n$. Thus we need alternative methods in this situation.

### *Regression in high-dimensions*

We can still apply *dimension reduction approaches* such as forward stepwise selection[53], ridge regression, the lasso, and principal components regression. These methods avoid overfitting data using a less flexible model.

[53] Backward selection can not be used here since we can not a fit the full model with all the predictors.

Figure 38 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 $n = 100$. 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

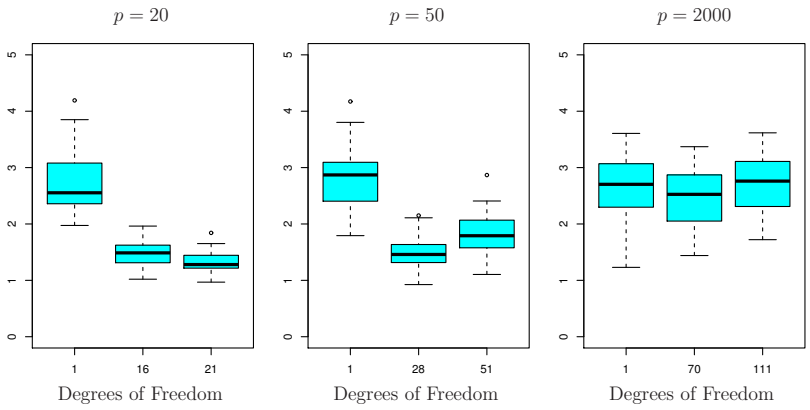



Figure 38: The lasso was performed with n = 100 observations and three values of p, the number of features. Of the $p$ features, 20 were associated with the response. The boxplots show the test MSEs that result using three different values of the tuning parameter $\lambda$. For ease of interpretation, rather than reporting $\lambda$, the degrees of freedom are reported; for the lasso this turns out to be simply the number of estimated non-zero coefficients. When $p = 20$, the lowest test MSE was obtained with the smallest amount of regularization. When $p = 50$, the lowest test MSE was achieved when there is a substantial amount of regularization. When $p = 2000$ the lasso performed poorly regardless of the amount of regularization, due to the fact that only 20 of the 2000 features truly are associated with the outcome.

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 $p + 1 \geq n$, 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 $p > n$, it is easy to obtain a useless model that has zero residuals. Therefore, one should never use sum of squared errors, p-values, $R^2$ 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 $R^2$ on an independent test set is a valid measure of model fit, but the MSE on the training set certainly is not.