Linear Regression Continued

Arnab Maity - Modified by Justin Post

Packages used in this set of notes:

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

# Methods for Selecting Variables & Evaluating Model Performance

Like any other learner, we need ways to evaluate the model performance and, in this case, determine which variables we want to include in our model. We can use

* inference-based methods
* model fitting criteria that do not require a test set
* training/test set ideas with a metric

Before we get into those, let’s first recap how we would use our fitted linear regression model to do prediction.

## Prediction

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

This is our prediction for two separate quantities:

* The mean response at this setting of predictors,
* A future response at this setting of predictors,

To visualize this, we can consider an SLR model. Recall our bike\_share data set. Here we fit a model using temperature to predict log\_rented\_bike\_count.

**Note: I’ve removed observations where functioning\_day was “No” as there were no bike rentals on these days. These were the set of points that always looked weird in our diagnostic plots!**

SLR\_fit <- lm(log\_rented\_bike\_count ~ temperature, data = bike\_share)  
summary(SLR\_fit)$coefficients |>  
 kable()

|  | Estimate | Std. Error | t value | Pr(>|t|) |
| --- | --- | --- | --- | --- |
| (Intercept) | 5.4005414 | 0.0152316 | 354.56279 | 0 |
| temperature | 0.0537668 | 0.0008657 | 62.11112 | 0 |

If we are interested in predicting the **mean rented bike count at a temperature of 22.22 degrees** (72 degree Fahrenheit), we’d plug 22.22 into our equation:

summary(SLR\_fit)$coefficients[1, 1] + summary(SLR\_fit)$coefficients[2, 1]\*22.22

[1] 6.595239

#or use predict()  
predict(SLR\_fit,   
 newdata = data.frame(temperature = 22.22))

1   
6.595239

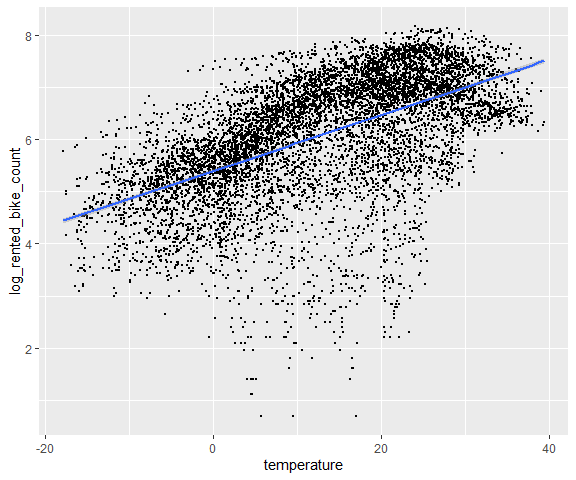
Likewise, if we wanted to predict a **future rented bike count at a temperature of 22.22 degrees**, we’d simply plug 22.22 into our equation!

The difference comes in the variability around the prediction. The variability associated with predicting a mean is generally far less than the variability associated with predicting a future observation!

Consider the scatterplot with an SLR fit below. If the graph is inspected very closely, one may notice a ‘confidence band’ around the line. This is a confidence interval that attempts to capture the **mean response** at a given temperature.

* With this type of interval, we are trying to capture the value of the line, across repeated samples, when the temperature is 22.22 degrees.

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



Scatterplot with fitted SLR model overlayed

Alternatively, we can try to capture a **new observation** when temperature is 22.22.

* We can produce a prediction interval using the predict() function in R and adding that to our graph

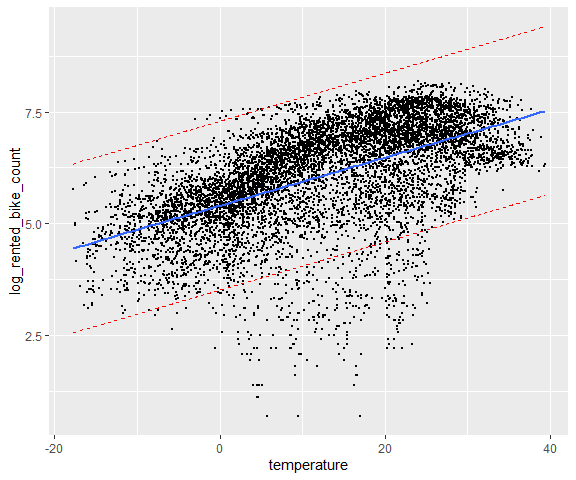
predictions <- predict(SLR\_fit,  
 newdata = bike\_share,  
 interval="prediction")  
predictions[1:4, 1:3]

fit lwr upr  
1 5.120954 3.230930 7.010978  
2 5.104824 3.214792 6.994856  
3 5.077941 3.187894 6.967987  
4 5.067187 3.177135 6.957239

bike\_share\_preds <- cbind(bike\_share, predictions)

* With this type of interval, we are trying to capture an observation about the line, across repeated samples, when the temperature is 22.22 degrees.

bike\_share |>  
 ggplot(aes(x = temperature, y = log\_rented\_bike\_count)) +  
 geom\_point(size = 0.5) +  
 geom\_smooth(method = "lm", se = FALSE) +   
 geom\_line(data = bike\_share\_preds,  
 aes(y = lwr),   
 color = "red",   
 linetype = "dashed") +   
 geom\_line(data = bike\_share\_preds,  
 aes(y = upr),   
 color = "red",   
 linetype = "dashed")



Scatterplot with fitted SLR model overlayed

While we produced confidence bands above, we can get these individual predictions using predict() in R:

conf\_for\_mean <- predict(SLR\_fit,   
 newdata = data.frame(temperature = 22.22),   
 interval = "confidence")  
conf\_for\_mean

fit lwr upr  
1 6.595239 6.569183 6.621295

pred\_for\_future <- predict(SLR\_fit,  
 data.frame(temperature = 22.22),   
 interval = "prediction")  
pred\_for\_future

fit lwr upr  
1 6.595239 4.705393 8.485085

We can interpret these intervals with statements such as

* when temperature = 22.22, we have confidence that the *mean* value of rented\_bike\_count will fall between (6.57, 6.62)
* when temperature = 22.22, we have confidence that a future value of rented\_bike\_count will fall between (4.71, 8.49).

# Model Selection

We now move into how to quantify model performance and, ultimately, how to choose which predictors, interactions, polynomial terms, etc. we should have in our model.

We’ve seen the use of hypothesis testing to understand the importance of predictors. We’ll investigate a few other options now:

* Comparing differing MLR models using CV or a train/test split
* Considering variable selection methods such as best subset selection, forward/backward selection, or combinations of these methods
  + With those methods, we can use p-values or other model performance metrics (, Adjusted , AIC, BIC, etc.) to choose the model form which don’t require a test set!
  + However, we could use the CV or train/test set idea to do these methods as well
* Utilizing penalized or regularized regression methods to choose fit our model
  + Some of these methods lead to automatic variable selection!
  + These methods require tuning of parameters
* Using dimension reduction techniques prior to model specification, or in conjunction with our model fitting

## Test Set Performance

As we discussed earlier, we can use the data splitting methods (CV, Bootstrap, holdout, etc.) to evaluate model performance on unseen test data.

This gives us a way to compare and choose between models when prediction performance is our major goal.

For example, the code below uses 5-fold CV, repeated 10 times, to estimate the test error for three competing models for predicting rented\_bike\_count. By repeating the CV process 10 times, we get a more stable estimate of prediction error.

* A model using only main effects.

set.seed(1001)  
# control params  
cv <- trainControl(method = "repeatedcv",   
 number = 5,   
 repeats = 10)  
# training main effects  
res\_main <- train(log\_rented\_bike\_count ~ hour + temperature + humidity +   
 wind\_speed + visibility + rainfall +   
 snowfall + seasons + holiday,   
 data = bike\_share,   
 method = "lm",   
 trControl = cv)

* A model with fewer variables with all main effects and their interactions.
  + In our R formula, we can pass (pred + pred2)^2 to do this concisely!

#training interaction model  
res\_interaction <- train(log\_rented\_bike\_count ~ (hour + temperature + wind\_speed + rainfall + snowfall + holiday)^2,   
 data = bike\_share,   
 method = "lm",   
 trControl = cv)

* Now a simpler model that may be more interpretable

#training simpler model  
res\_simple <- train(log\_rented\_bike\_count ~ hour + temperature + wind\_speed + rainfall + seasons,   
 data = bike\_share,   
 method = "lm",   
 trControl = cv)

Now we can investigate their repeated CV error to compare their fits.

rbind(c("Main effect", res\_main$results),  
 c("Interaction", res\_interaction$results),  
 c("Simple", res\_simple$results))

intercept RMSE Rsquared MAE RMSESD   
[1,] "Main effect" TRUE 0.7393104 0.596761 0.5339926 0.01806278  
[2,] "Interaction" TRUE 0.8165946 0.5102785 0.5895593 0.03353359  
[3,] "Simple" TRUE 0.8024988 0.5256906 0.5676996 0.02834004  
 RsquaredSD MAESD   
[1,] 0.01838994 0.00823254  
[2,] 0.03148973 0.01205185  
[3,] 0.02482149 0.01014892

The main effects only model wins here! (This won’t always be the case across problems you consider.)

## Metrics Used with Traditional Variable Selection Methods

In this section, we discuss methods to select a *subset* of the available covariates that we believe to be related to the response. We look at more traditional methods that don’t focus on predictive accuracy but use other model metrics or p-value based approaches to do so.

The final model selected through this method will be built by using least squares on the selected subset of variables.

### Metrics That Don’t Adjust for Complexity

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

* Residual squared error (RSE) (Not a good choice!)
* Coefficient of determination, (Not a good choice!)

These aren’t good choices. Let’s discuss them and then see why.

#### RSE

We have seen RSE as the estimator of in the previous sections. In general, RSE quantifies the uncertainty in prediction on from *even if the true regression parameters were known.*

We can view RSE as the amount the response will deviate on average from the true regression line.

A small RSE would indicate a good regression fit. In the bike\_share data example with only temperature as predictor of log\_rented\_bike\_count described above, we have .

Thus, even if we knew the true regression line (assuming that the linear model is correct), a prediction of log\_rented\_bike\_count based on temperature would still be off by units on average.

In the bike\_share data, the mean value of log\_rented\_bike\_count over all values of temperature is 6.09. Thus we are making an error in the amount of 16 percent.

The RSE is considered a measure of the **lack of fit** of the model.

* Small values of RSE imply the predictions are close to the observed values which indicate good model fit.
* Large values of RSE would indicate that the model did not fit the data well.

However, it is often not clear what values of RSE is acceptable.

The coefficient of determination () is another option to measure goodness of fit.

Coefficient of determination:

* Define the *total sum of squares (TSS)* as . Recall RSS is the residual sum of squares. Then

TSS measures the total variance in the response.

* We can think of TSS as the amount of variability inherent in the response before the regression is performed.
* RSS measures the amount of variability that is left unexplained after performing the regression.

Thus we can interpret as the *proportion of variance* in the response *explained by the model*.

* It can be shown that , with larger values indicting better fit.
* values close to zero would indicate that perhaps the linear model is wrong, and/or the error variance is high. Another way to interpret is that

#### Issues with RSE and

Usage of RSE and from the training set in model selection is **undesirable** as they will almost always choose the largest model possible.

That is, minimum RSE and maximum will almost always occur when number of predictors is largest.

* . will always decrease with the addition of more predictors.
* Likewise, will always increase with the addition of more predictors.
* To show this, let’s just do a silly example. We’ll fit an SLR model to the iris data with Petal.Length as a predictor and Sepal.Width as our response.

quick\_SLR <- lm(Petal.Width ~ Sepal.Length, data = iris)  
#RSE  
sigma(quick\_SLR)

[1] 0.4399958

#SSE  
(150-2)\*(sigma(quick\_SLR))^2

[1] 28.65225

#R^2  
cor(iris$Petal.Width,  
 quick\_SLR$fitted.values)

[1] 0.8179411

* Now we’ll add in a non-sense predictor that has nothing to do with Petal.Width

iris\_extra <- mutate(iris, nonsense = rnorm(150, sd = 3))  
#fit the model  
quick\_SLR\_2 <- lm(Petal.Width ~ Sepal.Length + nonsense,   
 data = iris\_extra)  
#RSE  
sigma(quick\_SLR\_2)

[1] 0.4413668

#SSE  
(150-3)\*(sigma(quick\_SLR\_2))^2

[1] 28.63628

#R^2  
cor(iris$Petal.Width,  
 quick\_SLR\_2$fitted.values)

[1] 0.8180539

We could use training test sets with these metrics but sometimes that is too computationally burdensome.

### Metrics That Adjust for Complexity

Alternatively, there are metrics available that adjusts training performance metrics to balance both goodness of fit and model complexity/size, so that a separate test 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 ,
* Information Criteria based metrics
  + Akaike information criterion (AIC)
  + Bayesian information criterion (BIC)
  + (Mallow’s) statistic.

Adjusted re-scales total sum of squares and RSS, before taking their ratio, to account for the number of predictors in the model.

In contrast, AIC, BIC and *add a penalty term* involving number of predictors to the training RSS to account for model size.

#### Adjusted

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

where is the number of predictors in the model. Maximizing the adjusted is equivalent to minimizing .

* Unlike , which monotonically decreases as increases, will increase and decrease as changes.
* We choose the model with maximum adjusted .

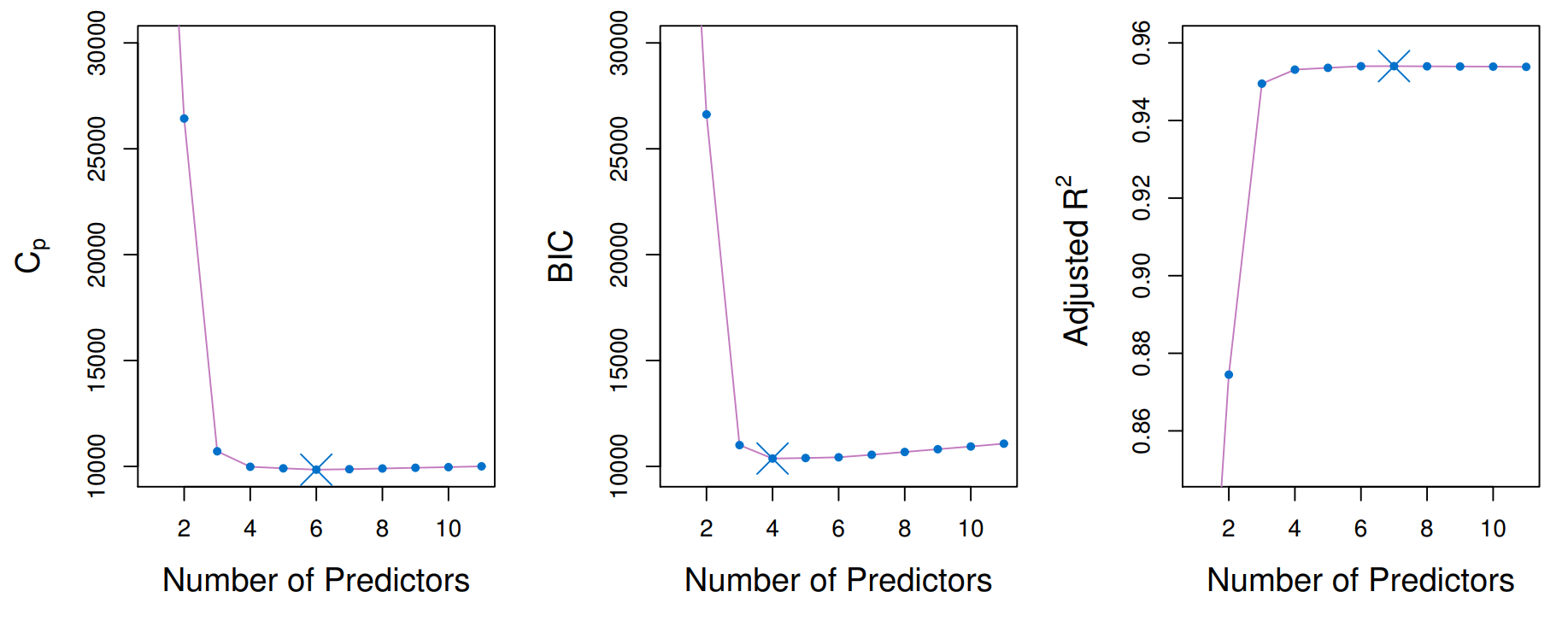
#### Information Criteria

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

where is a penalty term involving sample size, number of predictors in the model and estimated error variance using the full model containing all predictors.

The three metrics use the following form of :

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



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

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

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

## Traditional Variable Selection Methods

### Best subset selection

In this approach, we need to fit a separate least square model to *each* of the possible combination of the predictors in the dataset, that is, we need to fit all possible models.

* We can either use CV/holdout or AIC/BIC to choose the best model. The following algorithm shows the best subset selection procedure.

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

Note that we can use cross-validation for the entire set of possible models if we have such computational resources (for larger , this procedure can have tremendous computational burden). The algorithm above reduces this computational burden using Step 2, where it identifies the best model for each subset size *on the training set*.

Thus we reduce the problem from possible models to possible models. However, performing CV, if possible, has the distinct advantage over AIC/BIC that it directly estimates the test error for each model.

In R, we can use regsubsets() in the leaps package to perform best subset selection. We demonstrate this procedure using bike\_share data.

Note the usage of the argument nvmax = 11. This ensures that we will search of subsets up to size 11 (Since bike\_share data has 11 predictors - not including date).

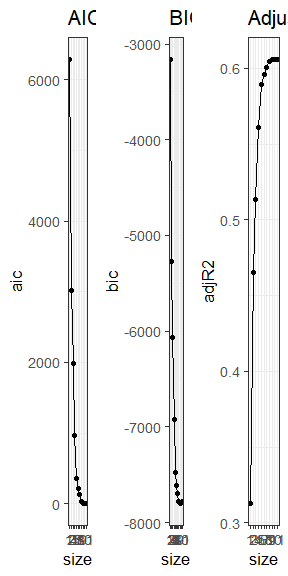
library(leaps)  
# Best model for each model size  
bestmod <- regsubsets(log\_rented\_bike\_count ~ hour + temperature + humidity + wind\_speed + visibility + dew\_point\_temperature + solar\_radiation + rainfall + snowfall + seasons + holiday,   
 data = bike\_share,  
 nvmax = 11)  
# summary  
mod\_summary <- summary(bestmod)

If you look at the summary() of our bestmod object we can see which predictors were included in the subset of each size. The output is not fun to look at though!

Now we can use either AIC/BIC or adjusted to choose the best model among these 11 models. We can pull of the model criterion for each model.

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

aic bic adjR2  
1 6285.654780 -3161.235 0.3130311  
2 3019.331357 -5273.141 0.4652219  
3 1986.736743 -6063.689 0.5133664  
4 970.208721 -6923.278 0.5607735  
5 361.851175 -7480.966 0.5891684  
6 220.201262 -7611.007 0.5958157  
7 124.010514 -7698.366 0.6003453  
8 30.796571 -7783.862 0.6047372  
9 10.418273 -7797.193 0.6057339  
10 9.111452 -7791.460 0.6058415  
11 10.209185 -7783.320 0.6058370



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

The minimum AIC and adjusted occurs for model size . For BIC it occurs for the model of size . These two fitted models are below.

#BIC best model  
round(coef(bestmod, 9), 3)

(Intercept) hour temperature   
 7.792 0.044 -0.028   
 humidity dew\_point\_temperature rainfall   
 -0.036 0.075 -0.227   
 seasonsSpring seasonsSummer seasonsWinter   
 -0.339 -0.307 -0.810   
 holidayNo Holiday   
 0.365

#AIC and adjusted R2 model  
round(coef(bestmod, 10), 3)

(Intercept) hour temperature   
 7.800 0.044 -0.027   
 humidity wind\_speed dew\_point\_temperature   
 -0.036 -0.015 0.074   
 rainfall seasonsSpring seasonsSummer   
 -0.227 -0.333 -0.304   
 seasonsWinter holidayNo Holiday   
 -0.805 0.364

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

### Forward Stepwise Selection

*Forward stepwise selection* considers a much smaller set of models as compared to best subset selection. The algorithm as as follows:

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

Forward stepwise selection involves fitting one intercept-only model, along with models in the th iteration, for . This reduces the computational complexity substantially from the best subset selection, which fits models for each .

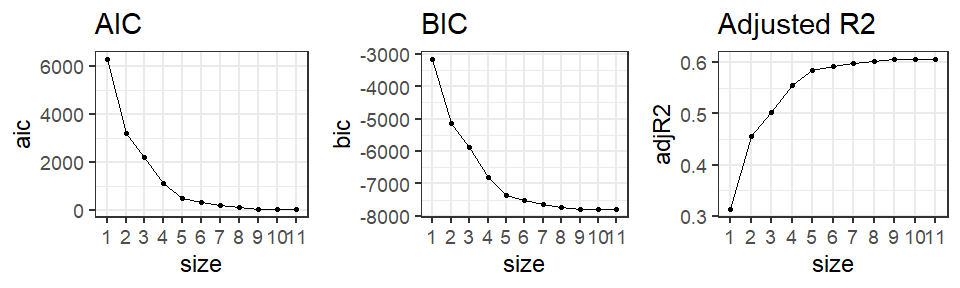
We should keep in mind that, since forward stepwise selection does not go through all possible models, there is no assurance that it will find the best model.

The following code performs forward stepwise selection for bike\_share data example.

forward <- regsubsets(log\_rented\_bike\_count ~ hour + temperature + humidity + wind\_speed + visibility + dew\_point\_temperature + solar\_radiation + rainfall + snowfall + seasons + holiday,   
 data = bike\_share,  
 nvmax = 11,  
 method = "forward")  
mod\_summary <- summary(forward)

As before, we could look at the summary() of this object but it isn’t nice to look at. Just as before, we can choose the best model among these 11 models by looking at our criteria.

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



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

In this case, we get the same exact results as with best subset selection (the same predictors are chosen in the models of size 9 and 10).

### Backward Stepwise Selection

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

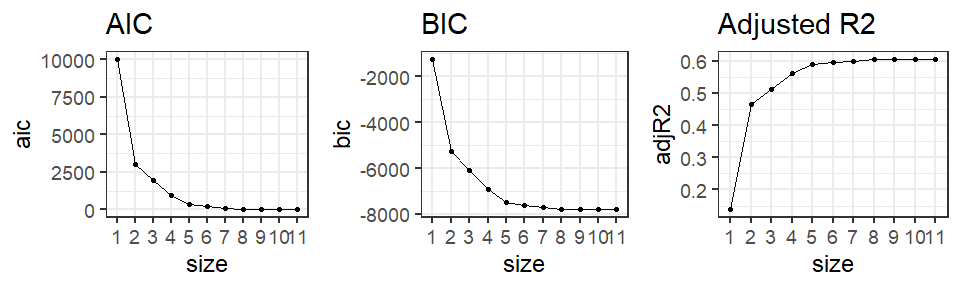
1. Start with the model with all the predictors included. Denote the model by .
2. For ,
   * consider all models that contain all but one of the predictors in , for a total of predictors.
   * choose the best among these models; denote this model by
3. Among the models , choose the best model using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

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

backward <- regsubsets(log\_rented\_bike\_count ~ hour + temperature + humidity + wind\_speed + visibility + dew\_point\_temperature +solar\_radiation + rainfall + snowfall + seasons + holiday,   
 data = bike\_share,  
 nvmax = 11,  
 method = "backward")  
# summary  
mod\_summary <- summary(backward)

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 11 models.

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



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

In the bike\_share data example seen so far, the results match regardless of method we used. This is generally not going to hold!

As the model chosen by AIC, adjusted , and BIC differ, we can pick the criteria we like the most (e.g. BIC for typically giving smaller models), and go with the corresponding best model.

### Using the Holdout and Cross-Validation for Subset Selection

As mentioned before, apart from AIC/BIC/adjusted , it is also possible to use data splitting techniques such as a holdout set or CV for model selection.

Ideally, we can run CV for each of the models, and choose the one with best test error. However, such an approach can be computationally expensive.

Alternatively, we can use the algorithms presented above and use CV on them. It is important to recall our discussion in the previous chapters about proper implementation of CV:

* the entire model building process, including any tuning, has to be applied to the training set.
* We **can not** simply use steps 1 and 2 on the full data to get and then just use CV on the final models.
* The following paragraph is quoted verbatim from the textbook to emphasize this important point (page 271).

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.

Thus we can think the **model size** as a tuning parameter here, since each training set might yield different models even if the size (number of predictors) 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 a style *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 the 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.

Let’s illustrate this.

* Obtain the train/test split.

set.seed(1001)  
## Create test and training sets  
data\_split <- createDataPartition(bike\_share$log\_rented\_bike\_count,   
 p = 0.8,   
 list = FALSE)  
  
test\_set <- bike\_share[-data\_split, ]  
train\_set <- bike\_share[data\_split, ]

* Apply best subsets on the training data

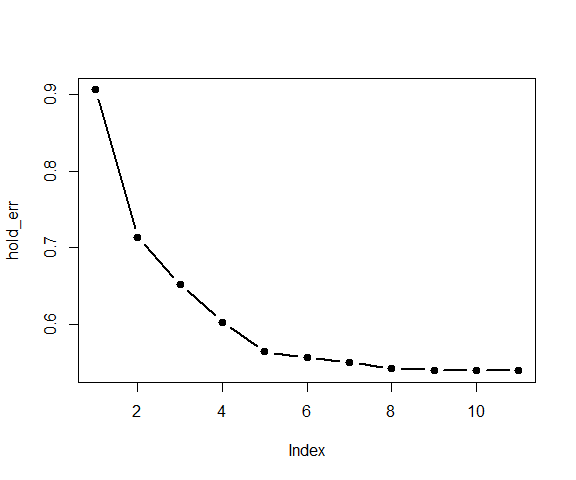
## Best subset selection on the training data  
best\_train <- regsubsets(log\_rented\_bike\_count ~ hour + temperature + humidity + wind\_speed + visibility + dew\_point\_temperature +solar\_radiation + rainfall + snowfall + seasons + holiday,  
 data = train\_set,  
 nvmax = 11)  
  
train\_sum <- summary(best\_train)

* For each model size, estimate the test performance
  + A function to help us

#We'll write a function to predict and estimate the error on the test set.   
#Inputs are   
#- model size (mod\_size),  
#- summary output 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, prediction, test error  
 model <- test\_model[, sub]  
 yhat <- model %\*% betahat  
 err <- mean((test\_resp - yhat)^2)  
 return(err)  
}

* Apply the function to each model size

#define the test model  
test\_model <- model.matrix(~ hour + temperature + humidity + wind\_speed + visibility + dew\_point\_temperature +solar\_radiation + rainfall + snowfall + seasons + holiday,  
 data = test\_set)  
  
#define the test response  
test\_resp <- test\_set$log\_rented\_bike\_count  
  
#apply the function to each of the model sizes  
hold\_err <- sapply(1:11, #apply the function to these   
 FUN = test\_err,   
 reg\_summary = train\_sum,  
 test\_model = test\_model,   
 test\_resp = test\_resp)  
  
#Let's plot the errors  
plot(hold\_err, type = 'b', pch=19, lwd=2)



* Choose the optimal model size and use that model size on a model fit to the full data set

size\_opt <- which.min(hold\_err)  
size\_opt

[1] 11

#fit on the full data set  
bestmod <- regsubsets(log\_rented\_bike\_count ~ hour + temperature + humidity + wind\_speed + visibility + dew\_point\_temperature +solar\_radiation + rainfall + snowfall + seasons + holiday,  
 data = bike\_share,  
 nvmax = 11)  
#Use the optimal size  
coef(bestmod, size\_opt)

(Intercept) hour temperature   
 7.857681e+00 4.398313e-02 -2.742862e-02   
 humidity wind\_speed visibility   
 -3.633705e-02 -1.502378e-02 -1.583471e-05   
dew\_point\_temperature rainfall seasonsSpring   
 7.495408e-02 -2.271213e-01 -3.383012e-01   
 seasonsSummer seasonsWinter holidayNo Holiday   
 -3.021923e-01 -8.112999e-01 3.630673e-01

In this particular example, we chose a 9-variable model. We refit to 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 9 variables (for this example), rather than simply using the variables that were obtained from the training set.
* This is because the best model with 9 predictors on the full data set may be different from the corresponding model on the training set.

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

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

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

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

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

Is this a correct application of cross-validation? Consider a scenario with N = 50 samples in two equal-sized classes, and p = 5000 quantitative predictors (standard Gaussian) that are independent of the class labels. The true (test) error rate of any classifier is 50%.

We carried out the above recipe, choosing in step (1) the 100 predictors having highest correlation with the class labels, and then using a 1-nearest neighbor classifier, based on just these 100 predictors, in step (2). Over 50 simulations from this setting, the average CV error rate was 3%. This is far lower than the true error rate of 50%.

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

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

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

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

## Regularization/Shrinkage Methods

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

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

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

### Ridge regression

Ridge regression shrinks the regression coefficients towards zero by imposing a *quadratic penalty* or penalty. The ridge regression coefficient estimates are obtained by minimizing

where is a tuning parameter. (Note that the intercept is not penalized.) The penalty term is called a *shrinkage penalty*. (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*.)

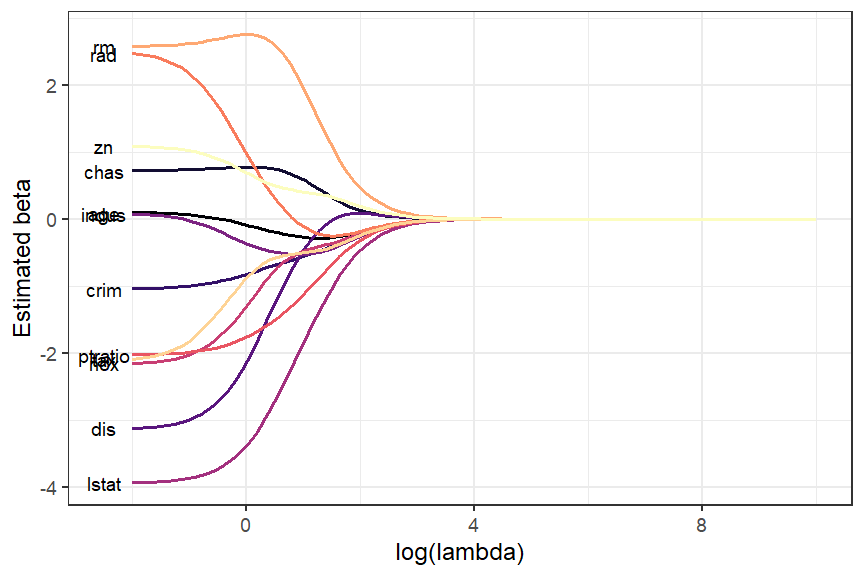
Here controls the relative impact of the two terms on the regression coefficient estimates. For large values of , the quadratic penalty term dominates the criterion, and the resulting estimates approach to zero. When , there is no penalty, and thus we get exactly the ordinary least squares estimates.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

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

Here denotes the identity matrix: a diagonal matrix with all diagonal elements being 1.

Notice again that setting gives us the least squares estimates, . Also note that, for , the matrix always has an inverse *even if* does not have full column rank. Thus, even in presence of collinearity/redundant columns in , ridge regression will still produce unique regression estimates. (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 shows the estimated ridge regression coefficients for different values of in Boston data with medv as response, and *standardized* predictors. The left most part of the plot corresponds to , and shows the least squares estimates. The right extreme of the plot represents a large value of , and we see that all the coefficients are very close to zero.



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

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

subject to the constraint

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

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

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

Before fitting the ridge regression model, we need to aware that scaling the predictors is often needed. In least squares estimation, scaling/standardizing a predictor does not *not* change the overall quality of the fit (e.g., , etc). If we multiply a predictor by a constant , then the resulting least square coefficient estimate will get multiplied by . In other words, using least squares, the quantity will remain the same no matter how we scale the -th predictor.[[1]](#footnote-60)

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

original scaled  
lstat -0.9500494 -0.1900099

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

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

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

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

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

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

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

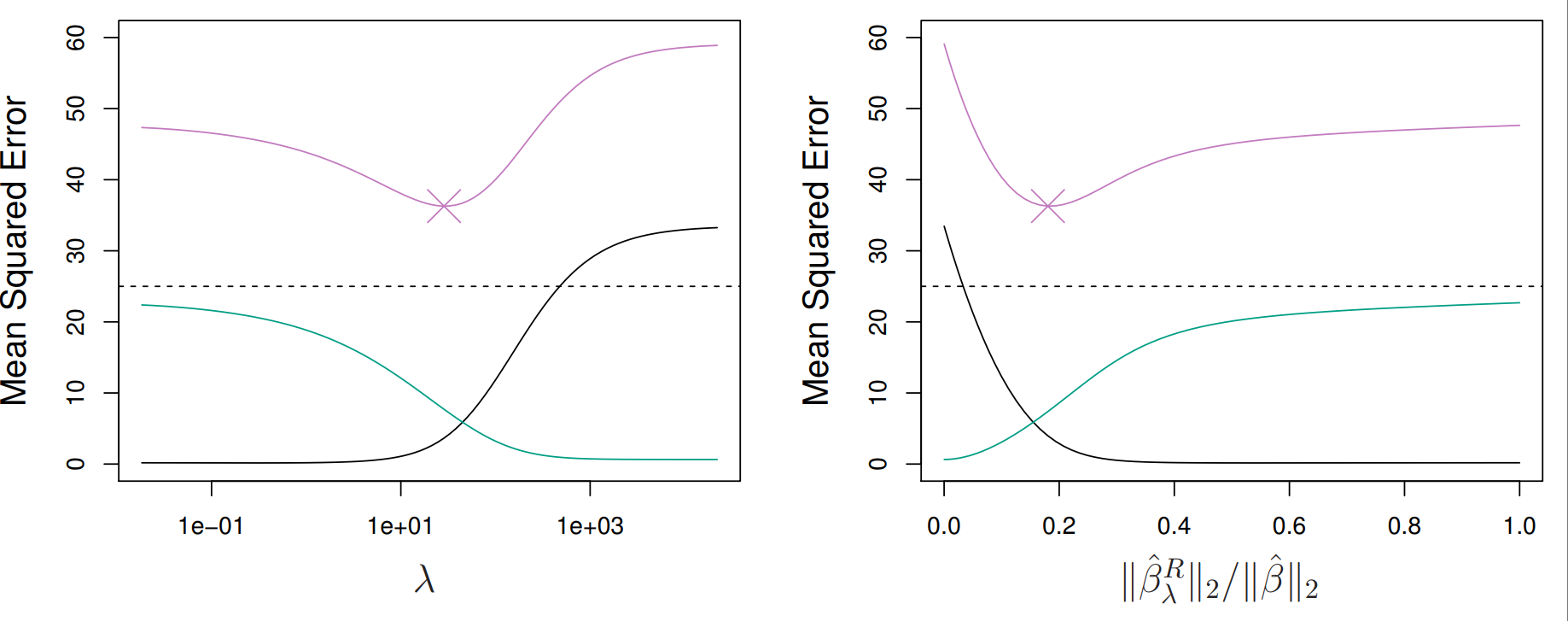
## Fit ridge regression for a grid of lambda  
grid <- 10^seq(-2, 10, length = 100)  
boston\_ridge <- glmnet(y = medv, x = model\_mat,  
 alpha = 0,  
 lambda = grid)  
betahat <- coef(boston\_ridge)

dim(betahat)

[1] 13 100

We constructed the model matrix by excluding intercept since it will be automatically included by glmnet() as well as excluding medv. Here we have used a custom grid of values.[[2]](#footnote-61) For each value of , the output betahat contains the corresponding estimates of the regression coefficients. Figure shows the estimated coefficients for different values of .

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



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

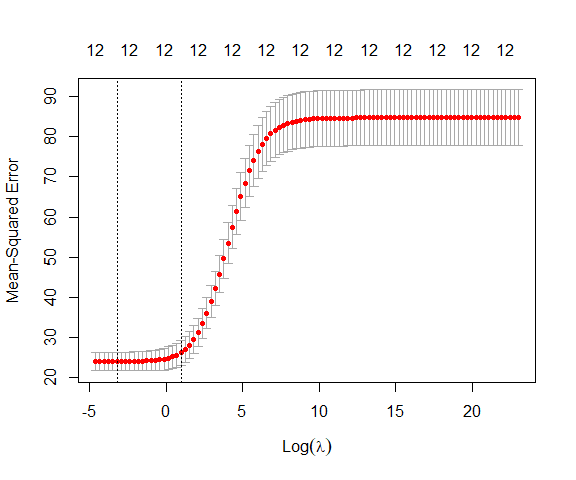
Ideally, we would like to select that minimizes test MSE. We can use data splitting methods such as cross-validation (or holdout) to do so. We choose a grid of candidate values of , and compute the cross-validation (or holdout) error for each value. The optimal is the one with minimum test error. Finally,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

We can use glmnet.cv() function to perform cross-validation. By default, glmnet.cv() uses 10-fold CV.[[3]](#footnote-65)

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

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

# Plot cv results  
plot(cv\_out)



Cross-validation results for Boston data using ridge regression.

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

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

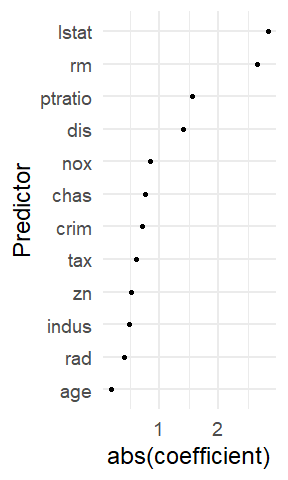
min one\_se  
1 0.04037017 2.656088

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

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

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

min 1se ols   
23.66276 23.02258 23.71326



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

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

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

### Lasso regression

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

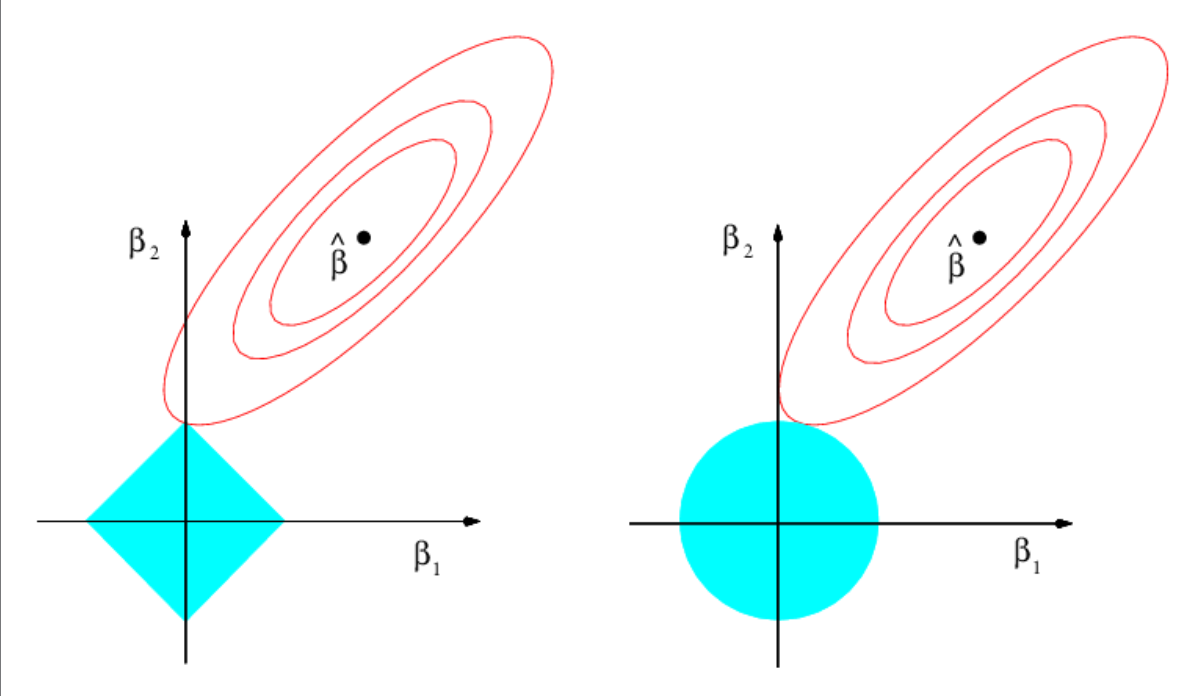
for . Due to the penalty term, there is no closed form solution to the lasso problem.[[4]](#footnote-73) An equivalent way to write the LASSO problem is in the form of a constrained minimization problem,

subject to the constraint

for some .

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

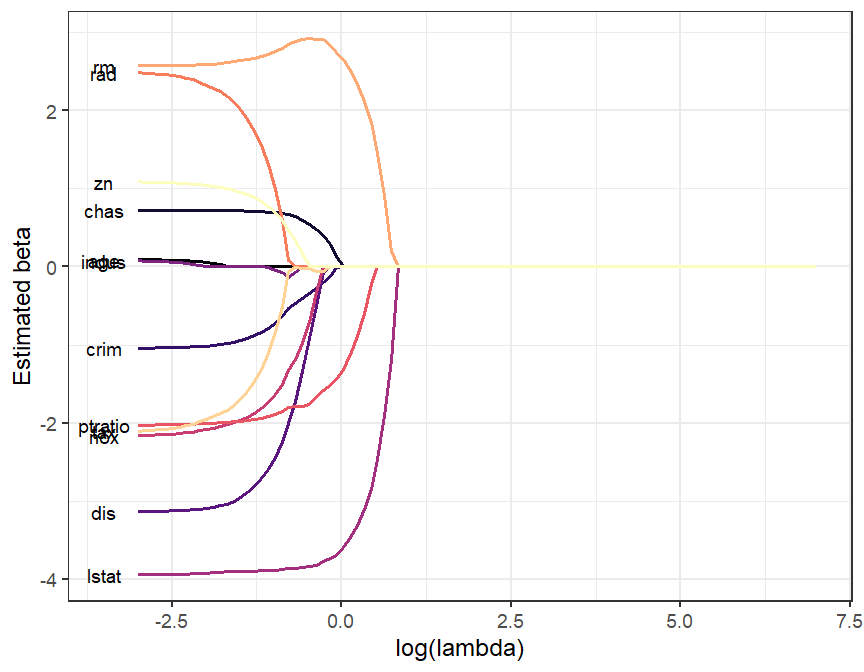
From the point of view of the constrained formulation, for large values of , we will effectively get the least squares estimates. Specifically, it can be shown that if is chosen larger that , then lasso estimates are identical to least squares estimates. On the other hand, if we chose , then the least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



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

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

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



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

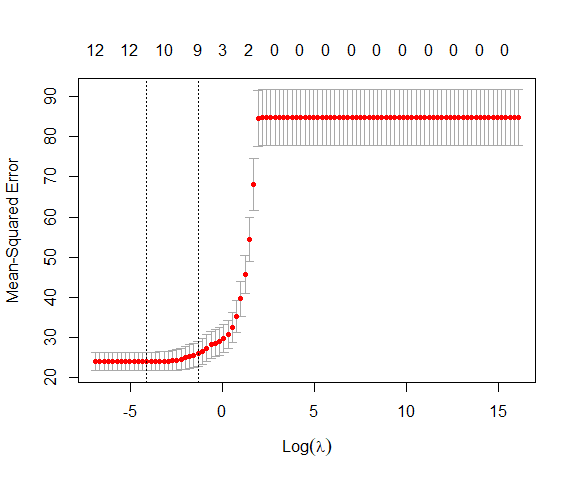
library(glmnet)  
## model matrix (standardized) and response  
medv <- Boston$medv  
model\_mat <- Boston[ , -13]  
model\_mat <- scale(model\_mat)  
model\_mat <- as.matrix(model\_mat)  
  
## Fit lasso regression for a grid of lambda  
grid <- 10^seq(-3, 7, length = 100)  
boston\_lasso <- glmnet(x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

[1] 13 100

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

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

# Plot cv results  
plot(cv\_out)

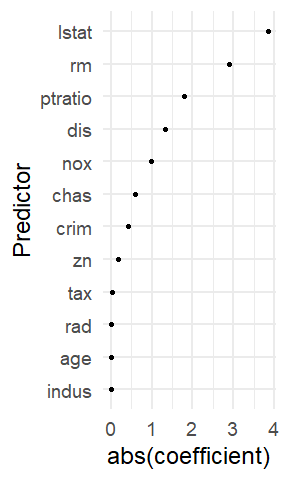


Cross-validation results for Boston data using lasso regression.

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

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

min one\_se  
1 0.01629751 0.2656088



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

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

betahat\_lasso

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

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

### Elastic net

A generalization of lasso and ridge is *elastic net*,[[5]](#footnote-87) which minimizes

for and . Note that lasso and ridge regressions are special cases of elastic net for and , respectively.[[6]](#footnote-88) 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.[[7]](#footnote-89) The elastic net penalty is a compromise between these two phenomena. The first term the the penalty encourages the correlated features to be averaged, while the second penalty term encourages sparsity in the estimated coefficients of the averaged features.

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

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

### Other variable selection methods

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

* *adaptive lasso*:[[8]](#footnote-91) 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*:[[9]](#footnote-92) 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*:[[10]](#footnote-93) does variable selection when the predictors have a natural ordering. For example, the predictors can be genes ordered by their chromosome location. Another example is when predictor is a function of time (functional data or time series). We can use the genlasso package here.
* *Smoothly clipped absolute deviations (SCAD)*[[11]](#footnote-94) and *Minimax concave penalty (MCP)*: produce sparse set of solution and approximately unbiased coefficients for large coefficients. Both methods are available in the ncvreg package.

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

## Dimension Reduction Methods

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

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

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

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

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

### Principal Components Regression

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

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

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

that explain most of the variability in the data.[[12]](#footnote-97) Typically we choose , and the new variables, , are ordered according to their importance. Specifically, is designed to capture the most variability in the original variables by any linear combination – this is called the *first principal component (PC)*. Then , the *second PC*, captures the most of the *remaining* variability while being *uncorrelated* to . We continue until we have the -th PC . In the end, we hope that the first few PCs, , will capture most of the variability in the original predictors.

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

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

[1] 506 12

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

[1] 29998.61

Before proceeding, let us check variances of individual predictors.

apply(X, 2, var)

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

Here we see an obvious problem – the variables are not comparable in terms of their variability. For example, the variable tax has a variance 2.8404759^{4} while lstat has variance 50.9947595. So majority of the total variation is due to tax. In such a case of imbalance, tax will overshadow all other variables. This may not be because tax is the only important variable here, but it is an issue of measurement unit/scale. For example, if we multiply lstat by , it does not make lstat any more important than it originally was, but its variance will be inflated by a factor of making lstat dominant over the rest of the predictors. To avoid this issue, we will standardize each predictor. [[13]](#footnote-98) Since now every predictor will have variance one, total variation is simply the number of predictor in the data.

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

[1] 12

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

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

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

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

[1] 506 12

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

sum(apply(Z, 2, var))

[1] 12

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

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

[1] 0.4922571

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

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

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

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

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

summary(pc\_out)

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

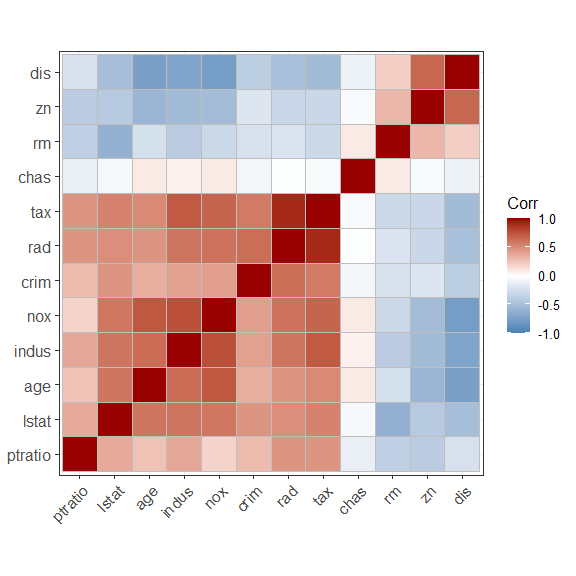
Note that we need all the 12 PCs to capture of , but doing so will not perform dimension reduction. Thus we will have discard the last few PCs and in the process sacrifice some of the variation in the original data. For example, if we are willing to sacrifice of TV (i.e., capture of TV), we will only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

Let us not briefly look at the *loadings* for the PCs.[[14]](#footnote-99)

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

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

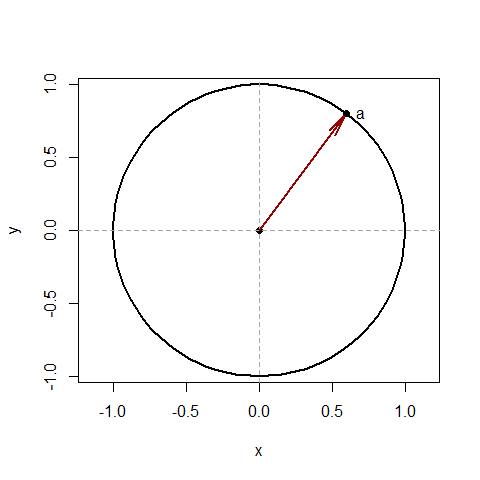
It seems PC1 has two groups of variables, (zn, rm and dis) vs. the rest of the variables excluding chas, with loadings with opposite signs but roughly similar magnitude. Investigation of correlation plot (Figure ) of the predictors gives insight about PC1 loadings. We can see there are two groups of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



Correlation plot of Boston data.

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

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



First PC direction.

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

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

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

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

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

Now that we have constructed the PCs, we can choose the first PCs, , and build a regression model with the PCs as predictors. Here we are assuming that the direction that the original predictors, show most variation are in fact the directions associated with the response.[[15]](#footnote-106)

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

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

In R, we can use the pcr() function in the pls package.[[16]](#footnote-107) Note the usage of arguments center = TRUE and scale = TRUE.

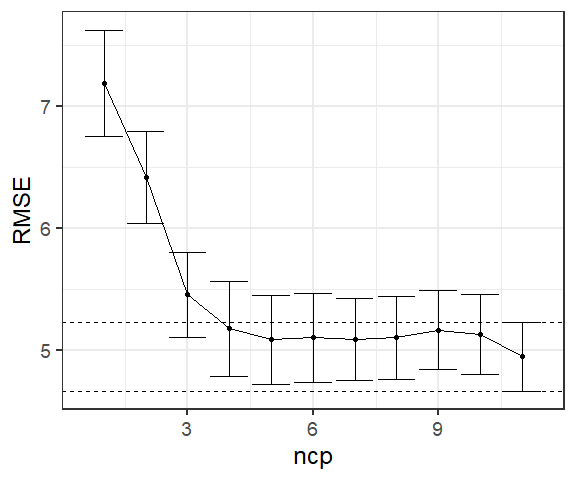
library(pls)  
set.seed(1001)  
pcr\_lm <- pcr(medv ~ .,   
 data = Boston,  
 center = TRUE, scale = TRUE,  
 validation = "CV")

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

summary(pcr\_lm)

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

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



Cross-validation error with one SE error bars.

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

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

model$results

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

Since we are using 10-fold CV, the standard error of the estimate of the test error (average of the 10 test errors) is simply the standard deviation divided by square-root of number of folds.[[17]](#footnote-111)

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

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

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

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

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

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

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

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

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

coef(pcr\_final)

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

Note that none of the original variables has zero coefficients.

### Partial Least Squares

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

In contrast, partial least squares (PLS)[[18]](#footnote-113) is a *supervised* approach, that is, PLS determines the linear combinations of the original predictors by making use of the response. Roughly speaking, the PLS approach attempts to find directions that help explain both the response and the predictors.

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

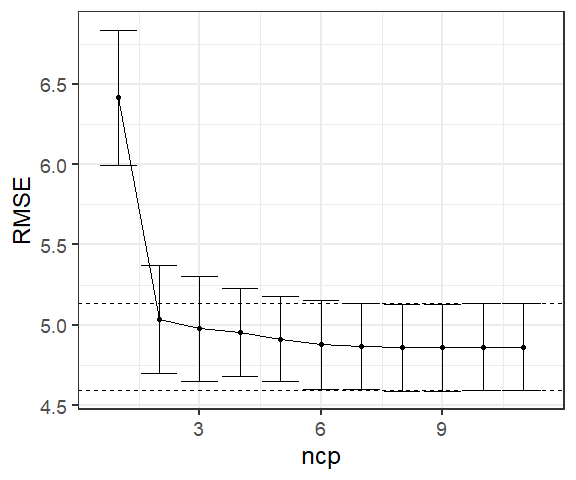
Thus the first PLS component places the highest weight on the variables that are most strongly related to the response.

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

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

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

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



Cross-validation error with one SE error bars.

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

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

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

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

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

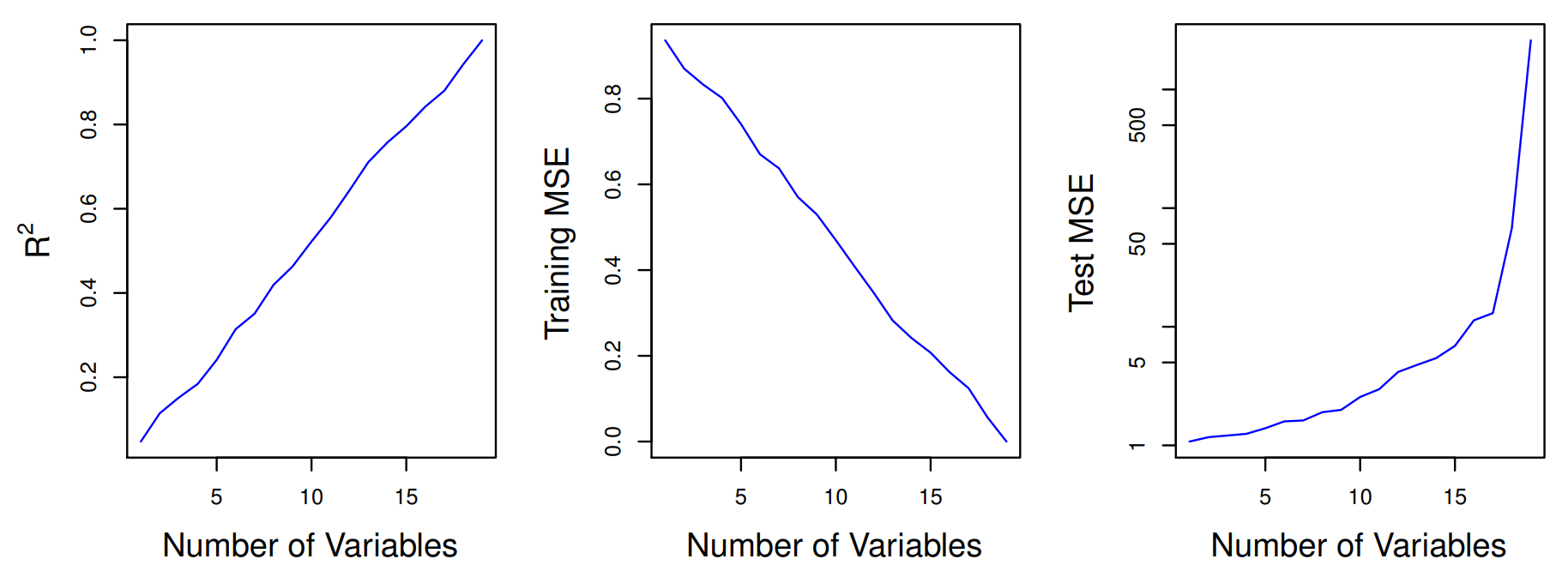
pls\_scores <- scores(pls\_final)  
load <- loadings(pls\_final)

It can be shown that PLS computes directions that have high variance and have high correlation with the response. In contrast, PCA seeks directions only with high variance.[[19]](#footnote-117) [[20]](#footnote-118) In practice PLS often produces performance similar to ridge regression or PCR. While the supervised dimension reduction of PLS can reduce bias, it also has the potential to increase variance.

# High-dimensional data

So far, all the methods we discussed assume that the number of predictors () is (much) less than the sample size (). The performance of these methods deteriorate as gets closer or exceed . Data sets containing more features than observations (or sometimes number of features slightly smaller than ) are often referred to as *high-dimensional*. In many fields, such as genomics and bioinformatics, such high-dimensional data are common. For example, in genomics we measure *single nucleotide polymorphisms (SNPs)*[[21]](#footnote-122) and investigate their association with an outcome of interest. Typically, the number of SNPs are in hundred of thousands, but sample size is in hundreds.

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

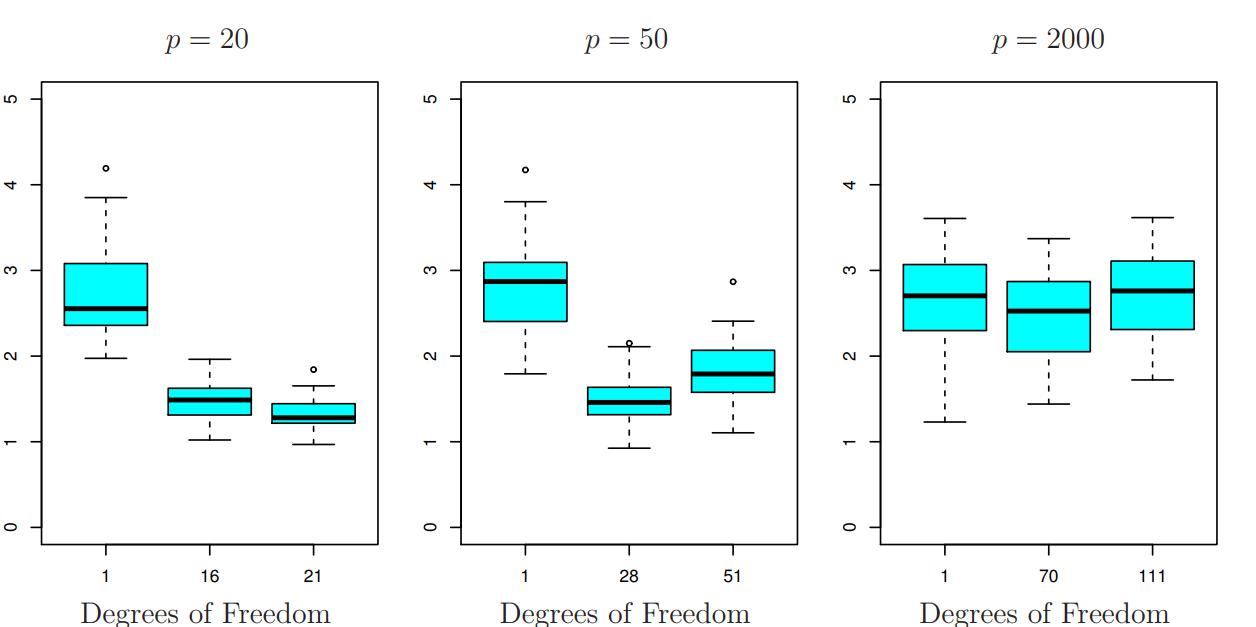


Risk of carelessly applying least squares when the number of features is high. Data were simulated with n = 20 observations, and regression was performed with between 1 and 20 features, each of which was completely unrelated to the response.

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

## Regression in high-dimensions

We can still apply *dimension reduction approaches* such as forward stepwise selection[[22]](#footnote-126), ridge regression, the lasso, and principal components regression. These methods avoid overfitting data using a less flexible model.



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

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

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

## Interpreting Results in High Dimensions

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

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

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

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

1. This is the reason we call least squares estimators *scale equivariant*. [↑](#footnote-ref-60)
2. glmnet() has a default way to set values as well if we do not specify manually. [↑](#footnote-ref-61)
3. See ?glmne.cv() for details. We can use nfolds argument to specify number of folds while using CV. [↑](#footnote-ref-65)
4. Computing the lasso solution is a . Efficient algorithms are available for computing the entire path of solutions as is varied. These algorithms have the same computational cost as for ridge regression. Interested readers should see for details. [↑](#footnote-ref-73)
5. Zou H, Hastie T (2005). Regularization and Variable Selection via the Elastic Net. Journal of the Royal Statistical Society, Series B, 67(2), 301–320. [↑](#footnote-ref-87)
6. This is the formulation glmnet() uses with the alpha argument. [↑](#footnote-ref-88)
7. See for more discussion. [↑](#footnote-ref-89)
8. Zou, H (2012). The Adaptive Lasso and Its Oracle Properties, JASA, 101, 1418 - 1429 [↑](#footnote-ref-91)
9. Yuan, M. & Lin, Y. (2007), Model selection and estimation in regression with grouped variables, Journal of the Royal Statistical Society, Series B 68(1), 49 - 67 [↑](#footnote-ref-92)
10. Tibshirani, R., Saunders, M., Rosset, S., Zhu, J. and Knight, K. (2005), “Sparsity and smoothness via the fused lasso”, Journal of the Royal Statistics Society: Series B 67(1), 91 - 108. [↑](#footnote-ref-93)
11. Fan J and Li R. (2001). Variable Selection via Nonconcave Penalized Likelihood and its Oracle Properties. Journal of American Statistical Association, 96:1348 - 1360. [↑](#footnote-ref-94)
12. Mathematically, we need to *normalize* the weights, that is, we ensure that . [↑](#footnote-ref-97)
13. This is my general recommendation when performing PCA. [↑](#footnote-ref-98)
14. We will discuss more about interpreting the loadings in a later chapter. [↑](#footnote-ref-99)
15. There is no assurance such an assumption actually holds. [↑](#footnote-ref-106)
16. Other packages such as caret can also so this. [↑](#footnote-ref-107)
17. Recall, for a random sample , standard error of sample mean is [↑](#footnote-ref-111)
18. Originally, Herman Wold developed the nonlinear iterative partial least squares (NIPALS) algorithm (Wold 1966, 1982) algorithm for nonlinear models. Later, Wold et al. (1983) adapted the NIPALS method for regression setting with correlated predictors – this adaptation was named PLS. [↑](#footnote-ref-113)
19. Stone M, Brooks R (1990). Continuum Regression: Cross-validated Sequentially Constructed Prediction Embracing Ordinary Least Squares, Partial Least Squares, and Principal Component Regression. Journal of the Royal Statistical Society, Series B, 52, 237 - 269. [↑](#footnote-ref-117)
20. Frank, I.E. and Friedman, J.H. (1993) An Statistical View of Some Chemometrics Regression Tools. Technometrics, 35, 109 - 135. [↑](#footnote-ref-118)
21. These are individual DNA mutations that are relatively common in the population [↑](#footnote-ref-122)
22. Backward selection can not be used here since we can not a fit the full model with all the predictors. [↑](#footnote-ref-126)