CS 598 Practical Statistical Learning

Alexandra Chronopoulou

2025-08-25

Contents

Course Information

Course Description

This course provides an introduction to modern techniques for statistical analysis of complex and massive data. Examples of these include model selection for regression, classification, nonparametric models such as splines and kernel models, regularization, dimension reduction, and clustering analysis. Applications are discussed as well as computation and theoretical foundations.

Course Prerequisites

Knowledge of basic multivariate calculus, statistical inference, and linear algebra. You should be comfortable with the following concepts: probability distribution functions, expectations, conditional distributions, likelihood functions, random samples, estimators, and linear regression models.

Course Learning Outcomes

By the end of the course, you will be able to:

- Use a broad range of methods and techniques in machine learning.
- Have a deeper understanding of major algorithms and techniques in machine learning.
- Build analytics pipelines for regression problems.
- Build analytics pipelines for classification problems.
- Build analytics pipelines for recommendation problems.

Textbook and Readings

There is no required textbook for this course.

Recommended resources for a statistical approach to machine learning are the following textbooks:

- An Introduction to Statistical Learning with Applications in R (basic)
- An Introduction to Statistical Learning with Applications in Python (basic)

- The Elements of Statistical Learning: Data Mining, Inference, and Prediction (more advanced)
- You may also want to view the Data School YouTube videos associated with the first book, as an additional resource.

The detailed syllabus for the course can be found on Coursera.

Chapter 1

Introduction to Statistical Learning

Statistical Learning is a realm in statistics, computer science, and data science focused on developing and understanding models that can *learn* from the data. While Statistics has long been dealing with model building, model validation, and prediction across a wide range of applications, the fairly recent increase in the volume, complexity and multidimensional nature of data has challenged traditional statistical techniques.

To this end, the introduction of sophisticated computer science algorithms, data management tools and optimization techniques has enabled us to design more powerful methods that are suited to handle the ever expanding nature of modern data.

Typically, *learning from data* means that given an **outcome/measurement** variable that may be quantitative or categorical, we are interested in making **predictions** based on a (typically large) set of features. More specifically, a *learner* or **prediction model** enables us to predict the outcome for **new unseen** objects.

We can separate statistical learning problems in two types:

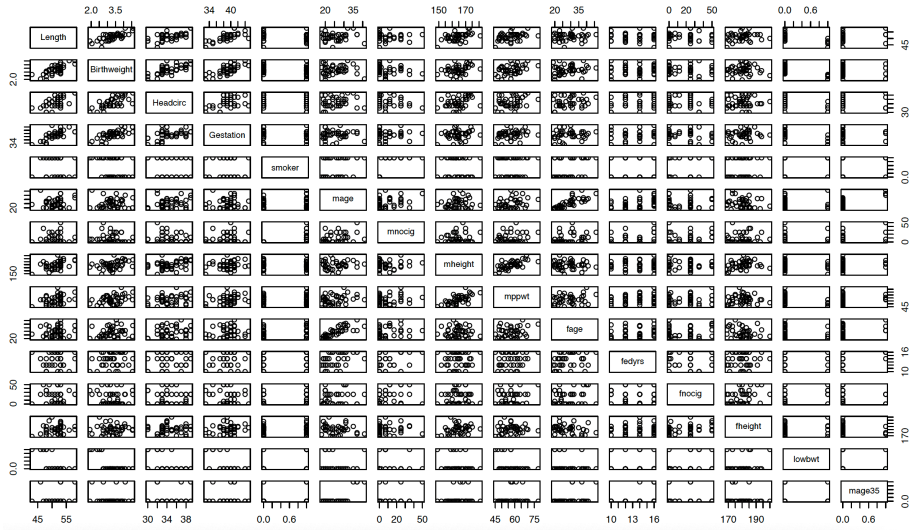
- **Supervised Learning**: when an outcome variable is present to guide the learning process. Examples of supervised learning are:
  - *Regression*: Response is quantitative, i.e. a *number.*
  - *Classification*: Response is qualitative, i.e. categorical, discrete, or a factor – a *label* (binary or multi-class).
- **Unsupervised Learning**: when an outcome variable is not present to guide the learning process. In this case, our goal is to identify latent structures in the data, e.g. clustering, association rules, HMM, etc.

1.1 Examples of Statistical Learning Problems

Birthweight Data (Regression)

This is an example in which the researchers are interested in predicting the weight of a newborn based on a set of baby and parent characteristics, such as gestation period, length, head circumferance, mother's and father's height, mother's weight, parents' smoking habits, etc. The study particularly focuses on babies born prematurely, i.e. before 40 weeks of gestation.

The scatterplots below illustrate the pairwise relationship of each variable with the other variables in the data set:



Note in the plots that three of the predictors, `smoker`, `lowbwt`, and `mage35` are categorical.

This is a typical example where a multiple linear regression model is a good starting point. If the fit of the model is good and all key model assumptions are satisfied, then the predictive power of this model will be strong.

Trees and Shrubs Data (Binary Classification)

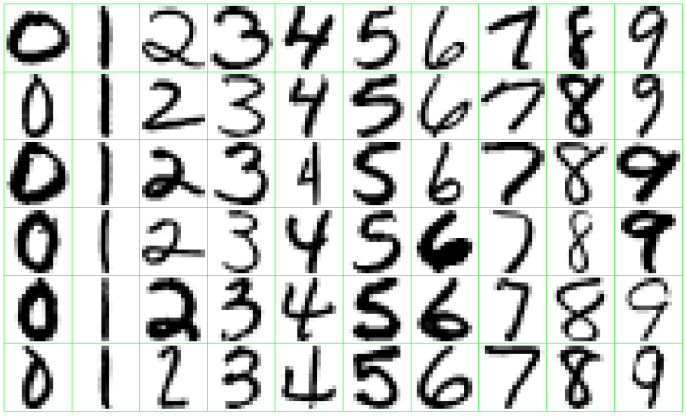
In this problem, the goal is to understand whether a specific woody plan, common in the Black Forest region in Southwestern Germany (Ref: Lederer) is a tree or a shrub. A sample of the data set is shown below:

Name	Height	Diameter	GrowthRate	Longevity
Silver Fir	55.0	1.50	0.5	500
European Beech	40.0	1.00	0.5	200
Common Hazel	5.0	0.10	0.5	70
Red Raspberry	1.5	0.01	0.4	10
European Spruce	50.0	1.20	0.5	600

This is a binary classification problem in which we want to use to use a set of characteristics to predict whether a specific plant is a tree or a shrub.

Handwritten Digits (Multiclass Classification)

This is a classic example of multiclass classification where the goal is to predict the handwritten digit $(0, \ldots, 9)$ from the envelope of a letter, based on a 16×16 eight-bit grayscale map (Ref: ESL). The challenge in this example is to keep the error rate below a desirable threshold to avoid misdirection of mail.



Proteomics Data (Clustering)

Proteomics is the field of biology and chemistry that is focused on studying proteins in large-scale. Typically, proteomics data sets are obtained as a result of an experiment and through specific processes, such as protein purification or through mass spectrometry, a technique that is used to measure mass-to-charge ratio of ions.

In the data set below, we show the expression matrix of 437 proteins only 101 of which are of interest (Ref: Romanova et al. – STAT 427 dataset). The challenge in these data sets is the amount of missing data (grey pixels in the plot) and the goal is to cluster the observations into various protein groups for further analysis.

1.2 Supervised Learning Framework

In this class, our main focus is *supervised learning*. This means that we have a **target/outcome/response** variable Y that we need to predict using a set of *features/predictors*, X. We can assume that the relationship between the response Y and the set of features are linked via a *parametric* function f as follows

$$\mathbf{y} \longleftarrow f(\mathbf{x}, w)$$

If we know the function f up to a parameter w, e.g. we know it is a linear, then our problem reduces to **learning** w.

For this reason, we collect (*training*) data $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^n$ that will help us to *minimize an* ***objective*** *function* with respect to w:

$$F(w) = \sum_{i=1}^{n} loss\Big(y_i,\ f(\mathbf{x}_i, w)\Big)$$

The *loss* function quantifies the distance between the model, i.e. $f(\mathbf{x}_i, w)$ and the respective outcome $\mathbf{y}_i$. It is up to the practitioner to choose the *loss* function. The most popular examples are:

- the Squared Error Loss

$$loss = \Big[y_i - f(x_i; w)\Big]^2$$

- the Absolute Error Loss

$$loss = \Big|y_i - f(x_i; w)\Big|$$

- the Log Loss

$$loss = \begin{cases} -\log f(x_i; w), & \text{if } y_i = 1 \\ -\log\Big(1 - f(x_i; w)\Big), & \text{if } y_i = 0 \end{cases}$$

- the 0-1 Loss

$$loss = \mathbf{1}_{\{y_i \neq f(x_i;w)\}} = \begin{cases} 1, & \text{if } y_i \neq f(x_i;w) \\ 0, & \text{if } y_i = f(x_i;w) \end{cases}$$

Depending on the choice of *loss* function, the minimizer of the objective function may or may not be in closed form. If not, we can try optimization algorithms that can guarantee convergence to the global minimizer. In the worst case scenario, one can always use gradient descent.

1.3 Why is Statistical Learning is Challenging

In our framework, we reduced the problem of learning, to estimating a vector of parameters. However, we assumed that we already know the true function f up to this parameter. In practice, we typically need to estimate or better *approximate* the function f. This approximation, denoted by $\hat{f}$ introduces an *additional source of error*.

Furthermore, in our framework we worked with a set of *training* data that will be used to learn the model. The objective function quantifies the distance between the *true* f parametrized by w and the observation in the training data set. However, to assess the actual performance and the predictive power of the fitted model, we need to *test* it to new, unobserved data. Thus, in the statistical learning context, our goal is to minimize the **test** or **generalization error**, not the training error.

Therefore, statistical learning is a difficult task because

- the training error typically underestimates the test/generalization error.
- the model performance might be good for training data, but poor for future (test) data due to overfitting.
- the number of parameters needed to learn the underlying regression or classification function f might be large; and in some cases larger than the available data.
- The test error increases significantly when the number of model parameters becomes larger.

Let's consider the following simple cases:

(1) In *classification*, the one-nearest-neighbor approach predicts perfectly on the training data, but poorly on testing data. An interesting illustration on how dimensionality changes the performance of linear classifiers can be found here: https://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification/

(2) On the other hand, in *regression*, if we take $\mathbf{y}_{n\times 1}$ to be the regression vector, where n is the sample size, and $\mathbf{X}_{n\times p}$ is the design matrix (i.e. matrix of predictors) with p predictors. The underlying regression model is

$$\mathbf{y}_{n\times 1} = \mathbf{X}_{n\times p}\, w_{p\times 1} + error$$

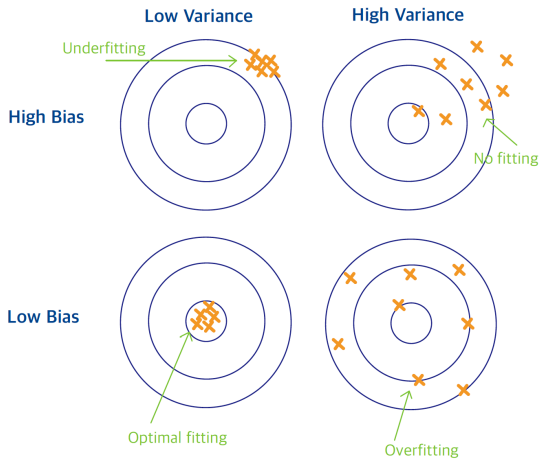
where $w_{p\times 1}$ are the model coefficients (i.e. parameters to estimate). If

- $p < n$: we have n (more) equations than the p parameters; this is when typically regression methods can be successful.
- $p = n$: we have n equations and n parameters; this leads to a perfect fit on training data.
- $p > n$: we have fewer equations n than parameters p: this is a scenario when classical regression methods do not perform well.

1.4 Bias-Variance Trade-Off

We mentioned before that we have **two sources of error** that affect the quality of the predictions we make. Specifically, we try to balance between:

- **Bias**: The bias is the difference between the estimated parameter or function and the true underlying parameter or function. High bias leads to under-fitting and an inaccurate model.
- **Variance** (of an estimated function): The variance quantifies the ability of the function to "adapt" to small changes in the data. High variance leads to over-fitting and an unreliable model.

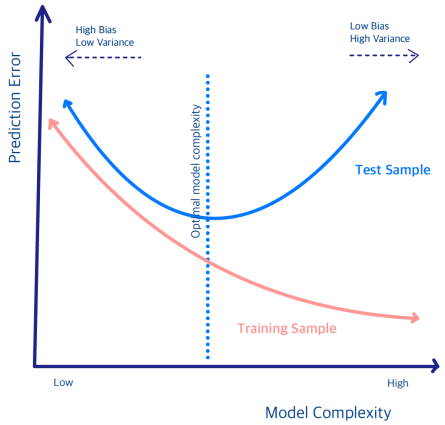


In general, one can think of our learning problem as a **target** in which the true model lies in the center. *How close to the center the fitted values are is captured by the bias.* If most of the points "miss" the target, then it means that our

model does not approximate the true f. If the bias is large, then increasing the sample size will not help us to "hit the target".

On the other hand, *the spread of the data is measured by the variance*. If all data points are all close to each other, then the variance is low, while if they are widely spread, then the variance is high.

Ideally, we would like to minimize both variance and bias at the same time. In our context, this is not feasible which means that we need to balance between the two. Keeping in mind that in statistical learning our goal is to *minimize the generalization error* –not the training error–, we can say that:



- When the number of model parameters is large, or in other words when the model is complex, the prediction error in the *training data* tends to **reduce**. However, when we overfit, we end up with a fitted model that follows precisely the pattern of the data and that won't generalize well, leading to a higher variance in the *testing data*.
- When the model is very simple, i.e. when we have very few parameters, then it is very likely that we underfit the data, leading to larger bias and a poor generalization error.

In this class, we will discuss:

(i) Flexible modeling techniques to reduce bias.

(ii) Useful strategies to achieve the trade-off between bias and variance.

As an example, two successful approaches that balance the bias-variance trade-off that we will study are.

- **Regularization**: Restrict the parameters to a low-dimensional space, which is adaptively determined by the data.
- **Ensemble**: Average many low-bias high-variance models $\longrightarrow$ averaging reduces variance.

1.5 Two Toy Examples: kNN vs. Linear Regression

Before we wrap up the introduction, we will review two simple supervised learning examples (i) k-Nearest Neighbors (kNN) (ii) Linear Regression
and will examine their performance and understand the bias-variance trade-off.

1.5.1 k-Nearest Neighbors

In the **k-Nearest Neighbors** (kNN) method, we use observations in the training set that are closest to $\mathbf{x}$ to form $\mathbf{y}$. Specifically, the k-Nearest Neighbor fit for $\hat{y}$ is

$$\hat{\mathbf{y}}(\mathbf{x}) = \frac{1}{k} \sum_{x_i \in N_k(\mathbf{x})} y_i$$

where $N_k(\mathbf{x})$ is the neighborhood of $\mathbf{x}$ defined by the k closest points x_i in the training sample. In a regression context the kNN fitted $\hat{\mathbf{y}}$ predicts $\mathbf{y}$ via a **local average**, while in the classification context kNN returns the **majority vote** in $N_k(\mathbf{x})$ or a probability calculated on the frequencies in $N_k(\mathbf{x})$. What can be challenging in the kNN approach is *tuning* k, the neighborhood size, and determining the metric to define the neighborhood.

- The choice of k is directly linked to the complexity of the method which is roughly equal to n/k.
  - When $k = 1$, the prediction at x_i is **exactly** y_i which means that we have zero training error.
  - When $k = n$, every neighborhood contains all the n training samples, so the prediction is the same no matter x.
- The default metric to define the neighborhood is the *Euclidean distance*:

  $$d(\mathbf{x}, \tilde{\mathbf{x}}) = \sum_{j=1}^{p} w_j (x_j - \tilde{x}_j)^2,$$

  where we would like to learn the w_j's from the data.

1.5.2 Linear Regression

Given a vector of inputs $\mathbf{x}^T = (x_1, x_2, ..., x_p)$, we **approximate** Y via a *linear* function

$$f(\mathbf{x}) \approx \beta_0 + \sum_{j=1}^{p} x_j \beta_j$$

Our goal is to *estimate* the parameters β_j using the Least-Squares (LS) method by minimizing the Residual Sum of Squares (*objective function*)

$$\min_{\beta_0, ..., \beta_p} \sum_{i=1}^{n} \Big(y_i - \beta_0 - x_{i1}\beta_1 - ... - x_{ip}\beta_p \Big)^2$$

The solution is easy to obtain (both in R/Python and analytically under certain assumptions) and the fitted value for the ith input x_i is given by

$$\widehat{y}_i = \widehat{y}(x_i) = x_i^T \widehat{\beta}$$

We leave the details to be discussed in Week 2.

Linear Regression in a Classification Context

We can apply linear regression on classification problems with $Y = 0$ or 1. In this case, we predict Y to be 1 if the LS prediction $f(x)$ is bigger than 0.5, and 0 otherwise. This approach has drawbacks. First of all, the squared difference $p(\mathbf{x}) = \Big(y_i - f(\mathbf{x}_i)\Big)^2$ is not a good evaluation metric, since considering a linear function $f(\mathbf{x})$ may result in values outside $[0, 1]$. Therefore, when we are in this context a Logistic regression is the gold standard according to which

$$\log \frac{p(\mathbf{x})}{1 - p(\mathbf{x})} \approx \beta_0 + \sum_{j=1}^{p} x_j \beta_j$$

More on this in Week 8.

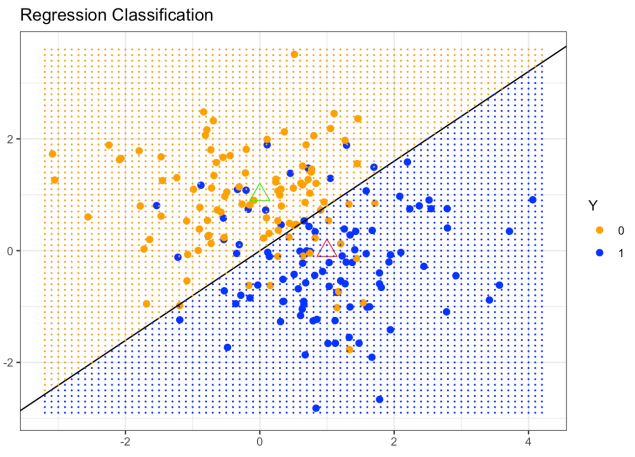
1.5.3 Simulated Binary Classification Example

Consider a response variable G that takes two values (0 – BLUE or 1 – ORANGE). In our simulation, we generate 200 such values; 100 in each class, and our goal is to use both regression and kNN to classify the data. The code behind the simulation and the plots can be found here.

We start by fitting a linear regression model to the simulated data. In a naive approach, we treat the response is a continuous variable. Hence, the continuous fitted values $\widehat{Y}$ are converted to a fitted class variable $\widehat{G}$ according to the following rule:

$$\widehat{G} = \begin{cases} \textit{Blue}, \text{ if } \widehat{Y} > 0.5 \\ \textit{Orange}, \text{ otherwise} \end{cases}$$

Our classification example is in two dimensions which means that the decision boundary (the boundary that separates the orange from the blue region) is a straight line.



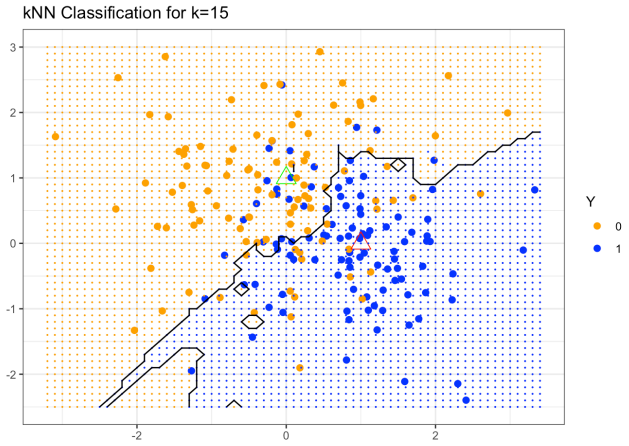
Specifically, the fitted decision boundary is a straight line (the black line in the plot) defined by $\mathbf{x}^T\hat{\beta} = 0.5$. Based on our simulation, we know that the blue region should be above the black fitted regression line, while the orange region should be below the black fitted regression line. We observe that there are many misclassifications on both sides of the decision boundary.

The regression line seems to be very smooth and too rigid when it comes to classifying the data. On the other end of the spectrum, we have the k Nearest-Neighbor approach. So, for the same simulated data, the nearest neighbor method will use the observations in the training set closest in input space to X to form $\hat{Y}$.

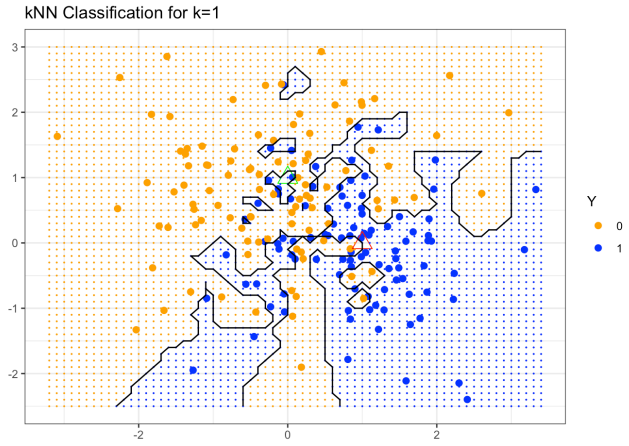
Using a 15-nearest-neighbor averaging of the binary coded response such that $\hat{Y}$ is the proportion of blue's in the neighborhood, then we assign

$$\hat{G} = \begin{cases} \textit{Blue}, \text{ if } \hat{Y} > 0.5 \\ \textit{Orange}, \text{ otherwise} \end{cases}$$

In this case, the predicted class is chosen by majority vote amont the 15 nearest neighbors.



We observe that the decision boundary separating the blue from the orange region is far more irregular than before and sensitive to local clusters of blue and orange dots. As a result, we have fewer misclassified observations than before. Remember that we can tune the neighborhood size. So, if we take the extreme scenario in which $k = 1$ and we only consider **one** neighbor, we have

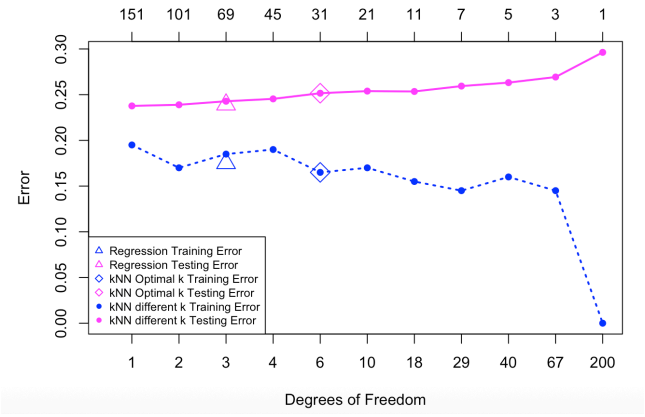


This results in a much rougher decision boundary with hardly any misclassified data. However, is this the ideal choice? The answer is no, but in order to understand the reason we need to consider the *generalization error*.

Up to now, we have used the same data for training and comparison purposes. As a result, a method like kNN has seemingly 0 error for $k = 1$. In order to make fair comparisons, we should consider another data set, independent of the one used to fit the data, so that we can compute the test/generalization error for both methods.

In the plot below, we compare the misclassification error on the testing data

set as a function of the degrees of freedom. In other words, we compare several kNN fits (for different ks) and the regression fit. The testing set here contains 10,000 observations.



The magenta curve is the test error and the blue curve is the training error for the kNN classification for different ks. In our simulated example, we used a 5-fold cross validation method to identify the optimal k. The results for the optimal k in kNN are denoted with a diamond. The results for linear regression are the magenta and blue triangles at 3 DFs (the DFs were determined based on the dimension of the linear model we fitted).

The regression method has the advantage of it being linear with only $p = 3$ parameters to estimate which means that it has a relatively *low variance*. However, the linearity assumption seems to be quite restrictive for the classification problem under consideration, which means that we expect to have high bias.

On the other hand, the kNN approach has no assumption of the shape of the underlying f, maybe except some local smoothness. This flexibility results in overfitting and a low bias (we saw that in the extreme case of $k = 1$).It can be shown that as $k, n \to \infty$ such that $k/n \to 0$, kNN is consistent. At the same time, the method has higher variance with the extreme case of when the number of parameters for kNN is roughly $n = k$, which goes to ∞ in order to achieve consistency.

1.5.4 Code for the Examples in the Lectures

We replicate one of the examples in the ESL book to illustrate the differences between the two simplest prediction methods for binary outcomes: the *Regression method* and the *k-Nearest-Neighbors method*.

1.5.4.1 Simulation

- The *generated data* consist of a **binary classification** with two classes, labeled as `0` and `1`.
- The *features* are two-dimensional.
- `Class 1` data points are generated from a Gaussian distribution with mean μ_1 and variance σ^2, i.e. $X_{1,1}, \ldots, X_{1,n_1} \sim \mathbb{N}(\mu_1, \sigma^2)$.
- `Class 0` data points are generated from a Gaussian distribution with mean μ_2 and variance σ^2,i.e. $X_{01}, \ldots, X_{0n_2} \sim \mathbb{N}(\mu_0, \sigma^2)$.
- In total, we generate *200 training samples* ($n = 100$ for each class), and we assign labels to the training data (100 `Class 1` and 100 `Class 0`).
- Similarly, we generate *10,000 test samples*.

1.5.4.2 Code for the Simulation

Set the model parameters for the simulation:

```
p = 2;            ## No. of parameters
sigma = 1;        ## St. Dev for the Normals (common)
mu1 = c(1, 0);    ## Vector of means for the first Normal
mu0 = c(0, 1);    ## Vector of means for the second Normal
```

Generate n i.i.d. (independent and identically distributed) samples from each normal to create the *training* data set.

```
n = 100;          ## Training Sample Size for each Normal

## rnorm(2*n*p) generates 2*n*p  N(0,1) random variables.
## matrix(rnorm(2*n*p), 2*n, p)*sigma generates a 2n-by-p matrix of two N(0, sigma^2) each of len
## Adding matrix(rep(mu1, n), nrow=n,byrow=TRUE) to each column of the previous matrix
##    shifts each of the columns to generate the Normals with means mu1 or mu0.
## Note that both mu0, mu1 are 2-dimensional.

traindata = matrix(rnorm(2*n*p), 2*n, p)*sigma +
            rbind(matrix(rep(mu1, n), nrow=n,byrow=TRUE),
            matrix(rep(mu0, n), nrow=n, byrow=TRUE))

# dim(traindata)

## We generate the 0 or 1 labels.
Ytrain = factor(c(rep(1,n), rep(0,n)))
```

Generate N *test* samples in a similar way:

```
N=10000;

testdata = matrix(rnorm(2*N*p), 2*N, p)*sigma+
           rbind(matrix(rep(mu1, N), nrow=N,byrow=TRUE),
```

```
            matrix(rep(mu0, N), nrow=N, byrow=TRUE))

Ytest = factor(c(rep(1,N), rep(0,N)))
```

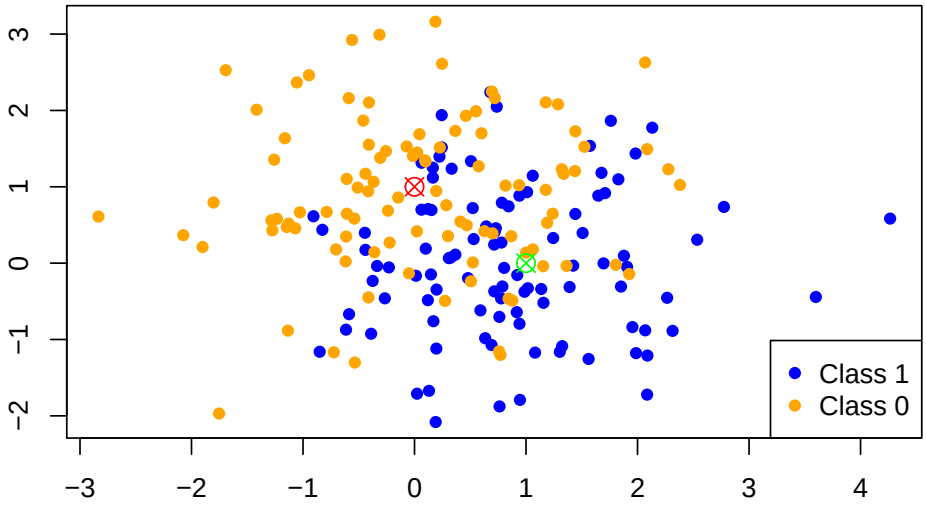
1.5.4.3 Visualization of the Simulated Data

This section also serves as a review of plotting in R.

We visualize the data we generated – those in the `traindata` matrix.

In the figure generated by the code below, points from two groups are colored in orange and blue, respectively; the two centers are plotted as +, and a legend is added to explain the association of each color.

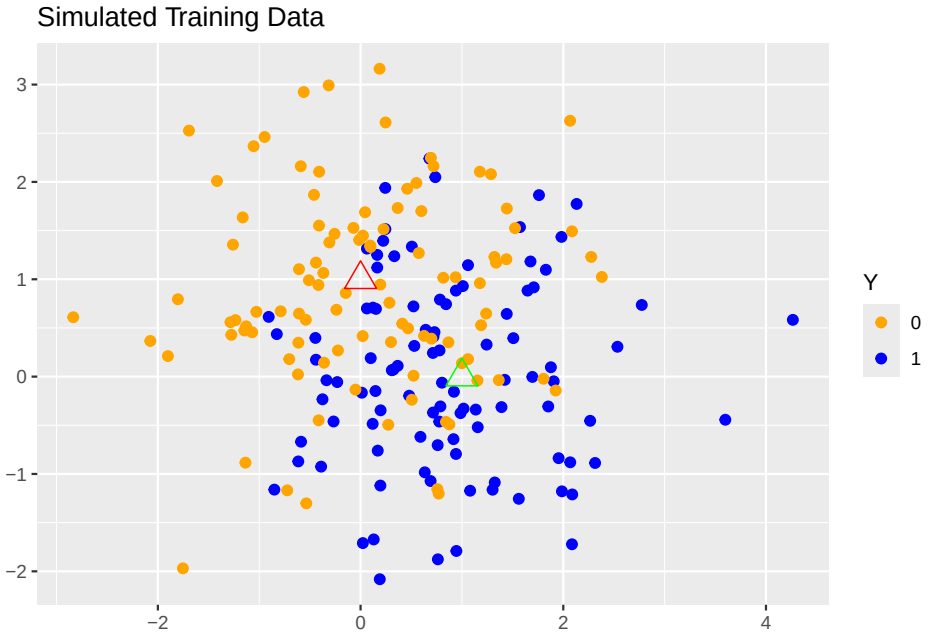
```
## Create an empty plotting area: The axes are the two vectors of normals generated,
## each one saved in a column of the `traindata` matrix.
## The following line creates an empty plot, since we used the option type="n"
## We do this so that we can color-code the data.
plot(traindata[,1], traindata[,2], type="n", xlab="", ylab="");

# Add the "Class 1" points - in blue color.
points(traindata[1:n, 1], traindata[1:n,2], pch=16, col="blue");

# Add the "Class 0" points - in orange color
points(traindata[(n+1):(2*n),1], traindata[(n+1):(2*n),2], pch=16, col="orange");

# Add the  centers for class 1
points(mu1[1], mu1[2], pch=13, cex=1.5, col="green");

# Add the  centers for class 0
points(mu0[1], mu0[2], pch=13, cex=1.5, col="red");

legend("bottomright", pch = c(16,16), col = c("blue", "orange"),
       legend = c("Class 1", "Class 0"))
```



1.5.4.5 Using the ggplot2 R library

In this section, we present an alternative way to construct the plot above via the `ggplot2` package. The `ggplot2` package allows us to create elaborate plots. More information can be found here: http://ggplot2.org/

```
# install.package("ggplot2")
library("ggplot2")

## The input in a `ggplot` function can only be a data.frame

## In our case the data is a matrix, so we convert them to data.frames here:
mytraindata = data.frame(X1=traindata[,1], X2=traindata[,2], Y=Ytrain)

## The ggplot output is an object which is saved in training.scatter.
## In this object, we can later add --and plot-- additional features.

training.scatter = ggplot(mytraindata, aes(X1, X2)) +  ## creates the empty plot
        geom_point(aes(colour=Y), size=2) + ## adds the points color-coded by the labels in Y
        scale_color_manual(values = c("orange", "blue")) +  ## change the default colors
        ## Use geom_point to add the centers (as before)
        geom_point(data=data.frame(X1=mu1[1], X2=mu1[2]), aes(X1, X2), colour="green", shape=2,si
        geom_point(data=data.frame(X1=mu0[1], X2=mu0[2]), aes(X1, X2), colour="red", shape=2, siz
        ggtitle("Simulated Training Data")  + ## add a title
        labs(x = "", y="")  ## remove axes labels

plot(training.scatter)
```



1.5.4.6 k Nearest-Neighbors Method

To apply the k-NN method, we need to choose k. In the example below, we use the neighborhood sizes suggested from the textbook (ESL). We also apply and plot the results for $k = 1$ and $k = 15$.

```
library("class")

neighbor_size = c(151, 101, 69,  45, 31, 21, 11, 7, 5, 3, 1); ## These are different k
m = length(neighbor_size);  ## needed to run our for-loop below.

train.err.knn = rep(0,m);   ## vector to store training error
test.err.knn = rep(0, m);   ## vector to store testing error

## knn is the R function that runs the kNN method. Output is a factor.

for( j in 1:m){
  Ytrain.pred = knn(traindata, traindata, Ytrain, k=neighbor_size[j])   ## predictions
  train.err.knn[j]= sum(Ytrain != Ytrain.pred)/(2*n)     ## mis-classification trainin
  Ytest.pred = knn(traindata, testdata, Ytrain,k=neighbor_size[j])      ## predictions
  test.err.knn[j] = sum(Ytest != Ytest.pred)/(2*N)       ## mis-classification testing
}

cbind(train.err.knn, test.err.knn)    ## matrix containing the train and test errors
```

```
##       train.err.knn test.err.knn
```

```
##  [1,]         0.285      0.24085
##  [2,]         0.270      0.24110
##  [3,]         0.265      0.24180
##  [4,]         0.280      0.24630
##  [5,]         0.280      0.25185
##  [6,]         0.305      0.25690
##  [7,]         0.255      0.27030
##  [8,]         0.235      0.27565
##  [9,]         0.220      0.28590
## [10,]         0.160      0.30210
## [11,]         0.000      0.31735
```

1.5.4.7 5-Fold Cross-Validation for Choosing optimal k

A systematic way to determine the *optimal* value for the neighborhood size, k, in a range of values is the so-called 5-fold Cross-Validation (CV) method. Essentially, the method selects the k value that minimizes the CV error. In a nutshell, the 5-fold CV error for each k is a *sum of 5 prediction errors*, one corresponding to each fold.

In the code below, we have an outside loop from 1 to 5 (the folds), and an inside loop from 1 to m (all possible values for k). Inside the loop, we use 80% (i.e., four folds) of the data as training and predict on the 20% (i.e., one fold) holdout set.

```
## In this chunk of code, we use the same vector of k's as above - the neighbor_size vector of si

## Initialize a vector cv.error to store the CV error for each k.
cv.error = rep(0,m);


id = sample(1:(2*n),(2*n), replace=FALSE);
fold = c(0,  40,  80, 120, 160, 200)

for(i in 1:5)
    for(j in 1:m){

      ## ith.fold = rows which are in the i-th fold
      ith.fold = id[(fold[i]+1):fold[i+1]];
      tmp = knn(traindata[-ith.fold,], traindata[ith.fold,], Ytrain[-ith.fold], k=neighbor_size[j
      cv.error[j]=cv.error[j] + sum(tmp != Ytrain[ith.fold])
    }

## Find the optimal k value based 5-fold CV
k.optimal = neighbor_size[order(cv.error)[1]]

## Error of KNN for the k is chosen by 5-fold CV
```

```
Ytrain.pred = knn(traindata, traindata, Ytrain, k=k.optimal)
train.err.knn.CV = sum(Ytrain != Ytrain.pred)/(2*n)
Ytest.pred = knn(traindata, testdata, Ytrain,k=k.optimal)
test.err.knn.CV = sum(Ytest != Ytest.pred)/(2*N)
```

1.5.4.8 Least Squares Method

We run a regression of `Ytrain` vs. the `traindata`, and we classify the results as follows:

$$\hat{Y} = \begin{cases} 1, \text{ if } fitted(Y) > 0.5 \\ 0, \text{ if } fitted(Y) \leq 0.5 \end{cases}$$

```
## Run a regression using the lm function
## Ytrain is a factor, so we need to convert it to a numeric vector to run the lm
RegModel = lm(as.numeric(Ytrain)-1 ~ traindata)

## Compute the \hat{Y} for training
Ytrain_pred_LS = as.numeric(RegModel$fitted > 0.5)

## Compute the predicted values for testing data and then the \hat{Y}
Ytest_pred_LS = RegModel$coef[1] + RegModel$coef[2] * testdata[,1] + RegModel$coef[3] *
Ytest_pred_LS = as.numeric(Ytest_pred_LS > 0.5 )


## Cross-tab for training data and training error
table(Ytrain, Ytrain_pred_LS);
```

```
##       Ytrain_pred_LS
## Ytrain  0  1
##      0 70 30
##      1 26 74
```

```
train.err.LS = sum(Ytrain !=  Ytrain_pred_LS) / (2*n);

## Cross-tab for test data and test error
table(Ytest, Ytest_pred_LS);
```

```
##      Ytest_pred_LS
## Ytest    0    1
##     0 7638 2362
##     1 2359 7641
```

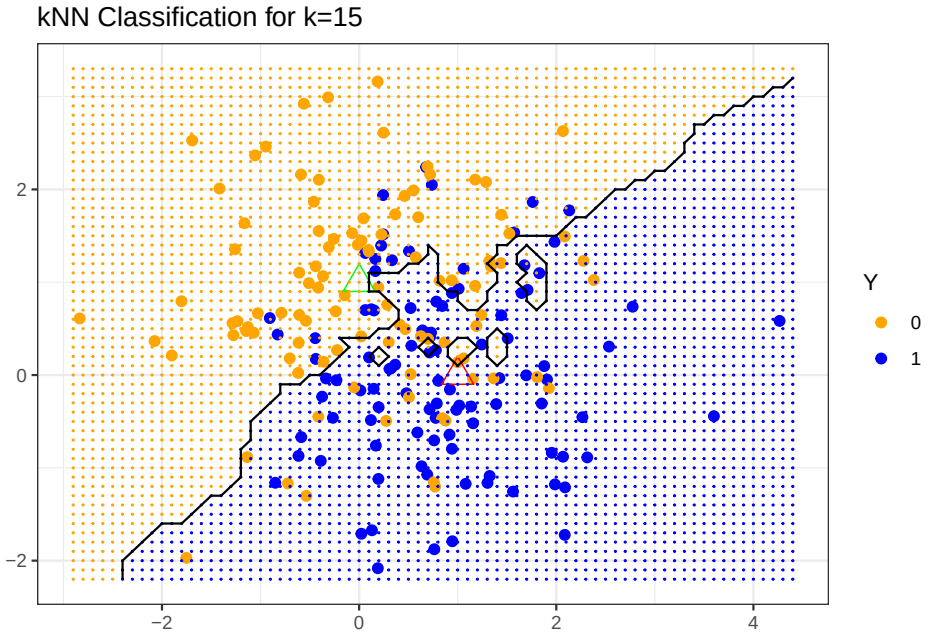
```
test.err.LS = sum(Ytest !=  Ytest_pred_LS) / (2*N);
```

1.5.4.9 Illustration of the Results

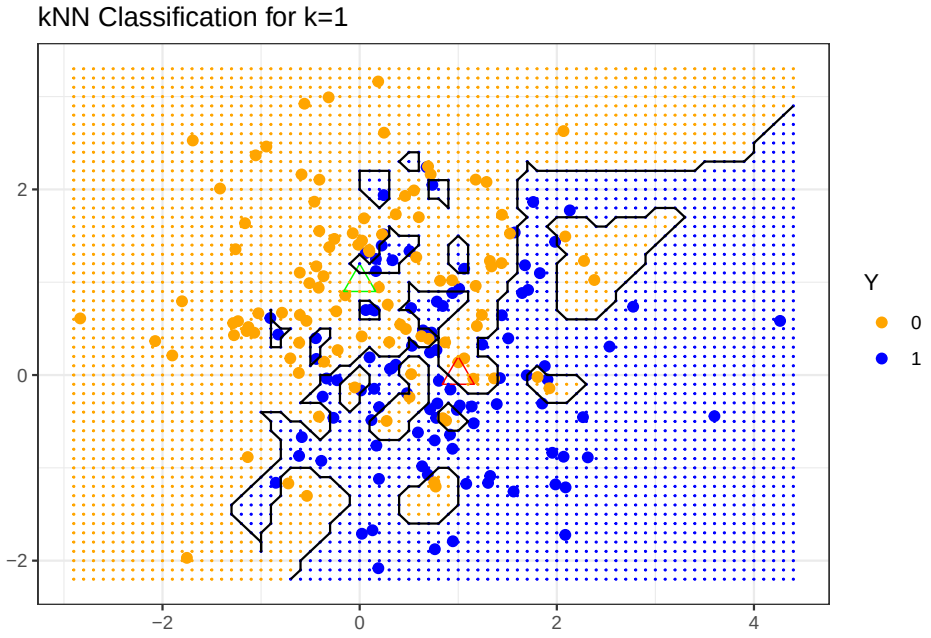
First, we illustrate the classification achieved by each method.

1.5.4.10 kNN Classification for two k's: k = 15 and k = 1

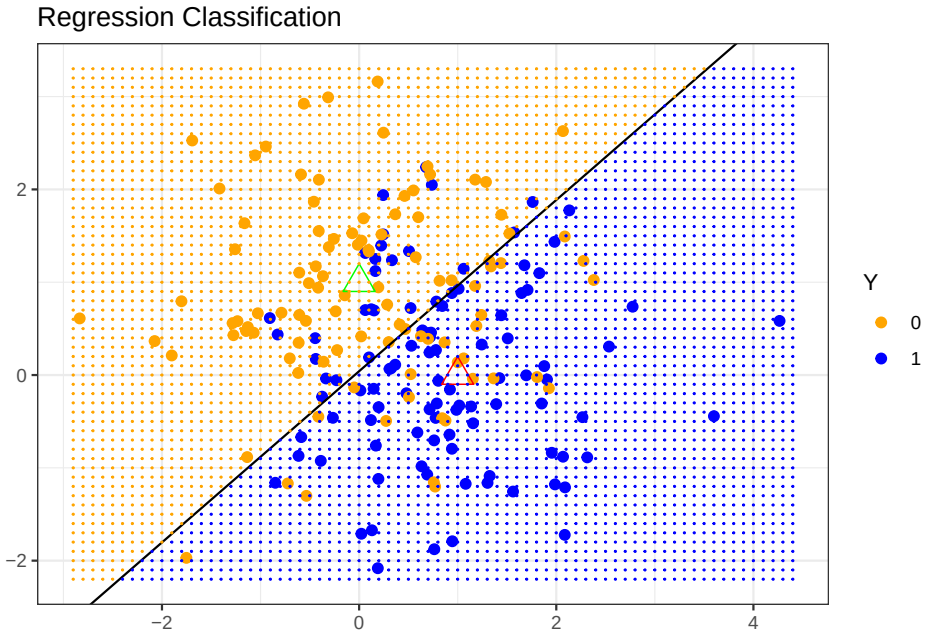
```
## Grid Using kNN Classification: We first define the boundaries for the grid

x.min = round(min(mytraindata$X1), digits=1)-0.1
x.max = round(max(mytraindata$X1), digits=1)+0.1
y.min = round(min(mytraindata$X2), digits=1)-0.1
y.max = round(max(mytraindata$X2), digits=1)+0.1
x.range = seq(from=x.min, to=x.max, by=0.1)
y.range = seq(from=y.min, to=y.max, by=0.1)
x.new = expand.grid(x.range, y.range)
names(x.new) = names(mytraindata[,-3])

## Basic scatterplot (same as before):
##   this is used as a base to add the boundary and shaded areas

grid.plot =  ggplot(mytraindata, aes(X1, X2)) +
             geom_point(aes(colour=Y), size=2) +
             scale_color_manual(values = c("orange", "blue"))   +
             geom_point(data=data.frame(X1=mu1[1], X2=mu1[2]), aes(X1, X2), colour="red", shape=2,
             geom_point(data=data.frame(X1=mu0[1], X2=mu0[2]), aes(X1, X2), colour="green", shape=
             labs(x = "", y="")


## Plot for k=15

knn.yhat.15 = knn(traindata, x.new, Ytrain,  k=15)
knn.pred.15 = ifelse(as.numeric(knn.yhat.15)>0.5,"1","0")

knn.plotdata.15 = as.data.frame(cbind(x.new, knn.yhat.15))


knn.plot.15 = grid.plot +
             theme_bw() + #remove grey background
             geom_point(data=knn.plotdata.15, aes(x=X1, y=X2, colour=knn.yhat.15), size=0.05) +
             geom_contour(data=knn.plotdata.15, aes(x=X1, y=X2, z=as.numeric(knn.yhat.15)), bins=1
             ggtitle("kNN Classification for k=15")

plot(knn.plot.15)
```




```
## Plot for k=1

knn.yhat.1 = knn(traindata, x.new, Ytrain,  k=1)
knn.pred.1 = ifelse(as.numeric(knn.yhat.1)>0.5, "1", "0")

knn.plotdata.1 = as.data.frame(cbind(x.new, knn.yhat.1))


knn.plot.1 = grid.plot +
             theme_bw() +
             geom_point(data=knn.plotdata.1, aes(x=X1, y=X2, colour=knn.yhat.1), size=0
             geom_contour(data=knn.plotdata.1, aes(x=X1, y=X2, z=as.numeric(knn.yhat.1)
             ggtitle("kNN Classification for k=1")

plot(knn.plot.1)
```




```
## Grid Using Linear Model Classification

RegModel = lm(as.numeric(Ytrain)-1 ~ traindata)
beta.hat = coef(RegModel)

## We use the same ranges for the grid as before

Reg.yhat = as.matrix(cbind(rep(1, nrow(x.new)), x.new)) %*% beta.hat   ## predicted Y values
Reg.pred = ifelse(Reg.yhat > 0.5, "1", "0")   ## convert numeric Y to factor

Reg.plot.data = cbind(x.new, Reg.pred)

## Basic plot is same as before, so now we add the regression boundary and classiffication result
reg.plot = grid.plot +
  theme_bw() +
  geom_abline(slope=-beta.hat[2]/beta.hat[3], intercept=(.5-beta.hat[1])/beta.hat[3],color="black
  geom_point(data = Reg.plot.data, aes(x=X1, y=X2, color=Reg.pred), size=0.05) +
  ggtitle("Regression Classification")

plot(reg.plot)
```



1.5.4.12 Plot the Performance and Compare the two Methods

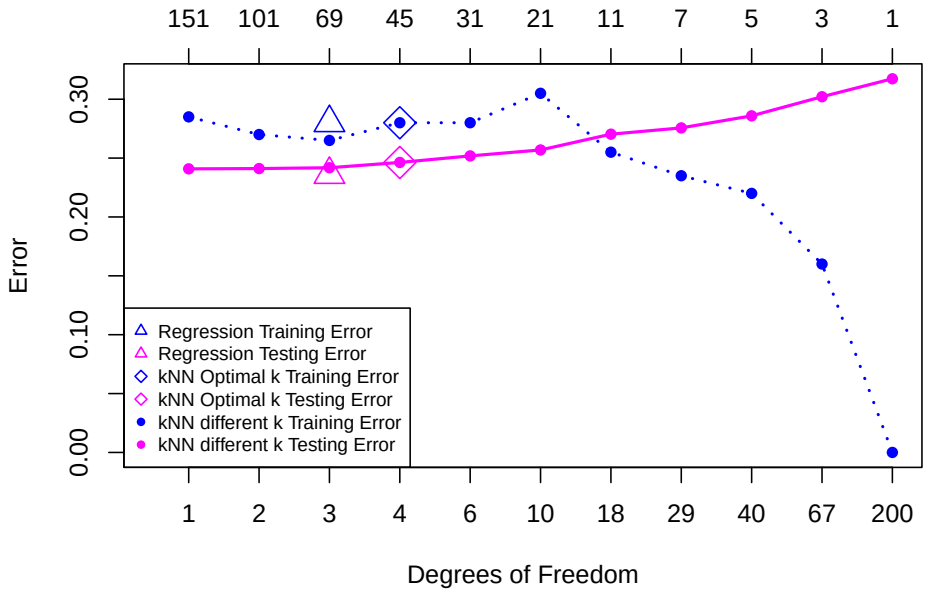
Test errors are in `magenta` and training errors are in `blue`. The upper x-coordinate indicates the k values, and the lower x-coordinate indicates the degrees-of-freedom of the kNN procedures so that the labels are reciprocally related to k.

The training and test errors for linear regression are plotted at $df = 3$ (corresponding to $k = (2n)/3$), since the linear model has 3 parameters, i.e., 3 dfs.

The training and test errors for KNN with k chosen by CV are plotted at the chose k values.

```
plot(c(0.5,m), range(test.err.LS, train.err.LS, test.err.knn, train.err.knn),
                  type="n", xlab="Degrees of Freedom", ylab="Error", xaxt="n")

df = round((2*n)/neighbor_size)
axis(1, at=1:m, labels=df)
axis(3, at=1:m, labels=neighbor_size)

points(1:m, test.err.knn, col="magenta", pch=16)
lines(1:m, test.err.knn, col="magenta", lty=1, lwd=2)
points(1:m, train.err.knn, col="blue", pch=16)
lines(1:m, train.err.knn, col="blue", lty=3, lwd=2)

points(3, train.err.LS, pch=2, cex=2, col="blue")
points(3, test.err.LS, pch=2, cex=2, col="magenta")
```

```
points((1:m)[neighbor_size == k.optimal], train.err.knn.CV, col="blue", pch=5, cex=2)
points((1:m)[neighbor_size == k.optimal], test.err.knn.CV, col="magenta", pch=5, cex=2)

legend("bottomleft", pch = c(2, 2, 5, 5, 16, 16),  col = c("blue", "magenta"),
       legend = c("Regression Training Error", "Regression Testing Error", "kNN Optimal k Trainin
```



Chapter 2

Linear Regression Models

A regression model is a model for the **conditional expectation** of the response/ outcome variable given a set of predictors/features. A linear regression model specifically, is *assumes* that $E(Y|\mathbf{X} = \mathbf{x})$ is a linear function of the inputs X_1, X_2, ..., X_p, i.e.

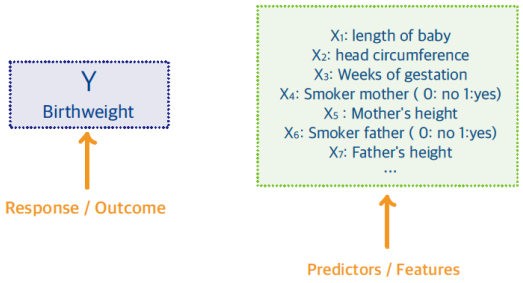
$$E(Y|\mathbf{X} = \mathbf{x}) \sim \beta_0 + \beta_1 X_1 + ... + \beta_p X_p$$

To describe the properties and characteristics of the MLR model, we introduce the `Birthweight` example.

Introduction to the `Birthweight` Example

The goal of the birthweight study is to **predict** the birthweight (*response/ outcome*) of a newborn baby given a set of *predictors/features*, specifically for babies born prematurely.

The features include baby-related characteristics, such as length, weight, head circumference, gestation period, and parent characteristics, such as mother's/father's height, smoking habits, mother's pre-pregnancy weight, mother's/father's age, etc.



The **predictors** we consider may be:

- *quantitative*, such as length, weight, gestation period, etc.
- *qualitative/categorical*, such as smoking habits (yes/no), or
- *transformations of quantitative inputs*: For example, power transformation of variables, e.g. X_3^3; continuous functions of variables, e.g. $\log X_5$; basic expansions leading to a polynomial representation , e.g. $\sum_{k=1}^{n} c_k X_{7k}$; interactions, e.g. $X_1 \cdot X_2$, etc.

2.1 Multiple Linear Regression (MLR) Model

The model formulation is given by:

Multiple Linear Regression Model

$$Y = \beta_0 + \beta_1 X_1 + ... + \beta_p X_p + \varepsilon$$

where

- β_0 is the intercept
- β_j is the regression coefficient associated with predictor X_j
- ε is the error term Usual assumptions for the error terms are $\varepsilon \sim IID(0, \sigma^2 \mathbf{I})$

Given the **training data** $\{x_{i1},\ x_{i2}, ..., x_{ip};\ y_i\}_{i=1}^{n}$, we can re-write the model as

$$y_i = \beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip} + \varepsilon_i,\ i = 1, ..., n$$

where n is the sample size. Using the following matrix representation for the response and the features:

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \quad = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix} \quad = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{pmatrix} \quad \mathbf{X} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{pmatrix}$$

the model equation can be written in as

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{pmatrix} \quad + \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}$$

which leads to the concise form

$$\underset{\text{Response}}{\mathbf{y}_{n\times 1}} = \underset{\text{Design Matrix}}{\mathbf{X}_{n\times(p+1)}} \underset{\text{Coefficients}}{\overset{(p+1)\times 1}{}} + \underset{\text{Error Term}}{\overset{n\times 1}{}}$$

where n is the sample size, and $p+1$ is number of predictors or columns of $\mathbf{X}$ plus the intercept (the "+1").

2.2 MLR Model Fitting

Given the training data $\{x_{i1},\ x_{i2}, \ldots, x_{ip};\ y_i\}_{i=1}^{n}$, we want to estimate , i.e. express:

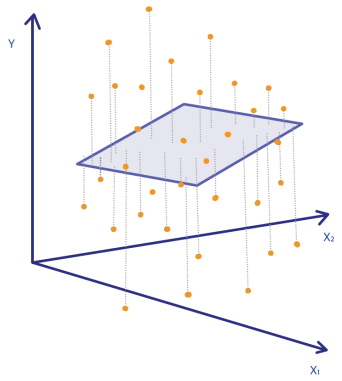
$$\hat{} = \left(\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p\right)^T$$

as a **function of the data**.

The estimated regression coefficients β are obtained by minimizing the *Residual Sum of Squares* (RSS):

$$RSS = ||\mathbf{y} - \mathbf{X}\beta||^2 = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta)$$

The RSS minimizes the (Euclidean) distance of the points from the regression surface:



This approach makes minimal assumptions, since it only requires that (x_i, y_i) are *random draws from their population*. Specifically, it makes **no assumptions** about the underlying distribution of the response. It simply finds the **best linear fit**, making no assumptions about model validity. In fact, the underlying (true) model does not even need to be linear for this approach to provide us with estimators. Therefore, it often gives good results, no matter how the data were obtained. If a linear model is a good approximation for the underlying non-linear (true) model, then the regression surface can also be thought as a criterion that measures the **lack-of-fit**.

Least-Squares Method

We want to estimate the **vector** of coefficients, by minimizing the *sum of squared residuals*:

$$RSS = ||y - \mathbf{X}\beta||^2 = (y - \mathbf{X}\beta)^T(y - \mathbf{X}\beta)$$

Therefore, we take derivatives with respect to β's and set to zero:

$$\frac{\partial RSS}{\partial \beta} = \mathbf{0}_{p\times 1} \Leftrightarrow$$
$$-2\ \mathbf{X}^T_{p\times n}(y - \mathbf{X}\beta)_{n\times 1} = \mathbf{0}_{p\times 1}$$

This leads to the so-called **Normal Equations**

$$\mathbf{X}^T(y - \mathbf{X}\beta) = \mathbf{0}$$

Solving the Normal Equations

$$(\mathbf{X}^T\mathbf{X})\ \beta = \mathbf{X}^T\ y$$

leads to the

Least Squares Estimators in MLR

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\ y$$

We assume that **the rank of X is** $p + 1$, i.e. no columns of $\mathbf{X}$ are a linear combinations of the other columns of $\mathbf{X}$. *Since* $\mathbf{X}$ *has rank* p*, the inverse of* $(\mathbf{X}^T\mathbf{X})$ *exists.*

Fitted, Predicted Values & Residuals

The **fitted value** of y_i at $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$ is computed as:

$$\hat{y_i} = \hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \hat{\beta}_2 x_{i2} + \ldots + \hat{\beta}_p x_{ip}$$

More generally, using matrix formulation, we can compute the **fitted values** of $\mathbf{y}$ based on the model as follows:

$$\begin{aligned}\hat{\mathbf{y}}_{n\times 1} &= \mathbf{X}\hat{\beta} \\ &= \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\ y \\ &= \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\ y := \mathbf{H}_{n\times n}y_{n\times 1}\end{aligned}$$

The Hat Matrix

We define

$$\mathbf{H}_{n\times n} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$$

to be the **hat matrix**, since it returns the "*y-hat*" values.

The **predicted** value of y_i at $x_i^* = (x_{i1}^*, x_{i2}^*, \dots, x_{ip}^*)$ is given by

$$\hat{\mathbf{y}}^* = \hat{\beta}_0 + \hat{\beta}_1 x_{i1}^* + \hat{\beta}_2 x_{i2}^* + \dots + \hat{\beta}_p x_{ip}^*$$

More generally, using matrix formulation, the **predicted values** of $\mathbf{y}$ are given by

$$\hat{\mathbf{y}}^* = \mathbf{X}^* \hat{\beta}$$

> Note that the difference between the fitted and predicted values is that the $\mathbf{x}$ is a vector of features that we have already observed (it is part of our training data), while $\mathbf{x}^*$ is an **unobserved** vector of features that is **independent** of the training data.

The **residual** of y_i at $x_i = (x_{i1}, x_{i2}, \dots, x_{ip})$ is obtained by

$$\begin{aligned} r_i &= y_i - \hat{y}_i \\ &= y_i - \hat{\beta}_0 - \hat{\beta}_1 x_{i1} - \hat{\beta}_2 x_{i2} - \dots - \hat{\beta}_p x_{ip} \end{aligned}$$

Using matrix formulation, the residuals can be computed as

$$\begin{aligned} \mathbf{r}_{n\times 1} &= \mathbf{y} - \hat{\mathbf{y}} \\ &= \mathbf{y} - \mathbf{X}\hat{\beta} = \mathbf{y} - \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\, y \\ &= \mathbf{y} - \mathbf{H}\mathbf{y} = (\mathbf{I} - \mathbf{H})\mathbf{y} \end{aligned}$$

The residuals $\mathbf{r}$ are used to estimate the **error variance**:

$$\hat{\sigma}^2 = \frac{1}{n-p-1}\sum_i r_i^2 = \frac{RSS}{n-p-1}$$

Note that the denominator in the formula is equal to the *degrees of freedom of the residuals* $(n-p-1)$.

Properties of the Residuals

The LS estimator is the β vector that satisfies the **normal equations**, that is

$$\mathbf{X}^T(\mathbf{y} - \hat{\mathbf{y}}) = \mathbf{X}^T(\mathbf{y} - \mathbf{X}\hat{\beta}) = \mathbf{0}$$

Based on this, we can derive the following properties for the residuals $\mathbf{r}_{n\times 1}$:

- The cross-products between the residual vector $\mathbf{r}$ and *each column of* $\mathbf{X}$ are zero, i.e.

$$\begin{aligned} \mathbf{X}^T\,\mathbf{r} &= \mathbf{X}^T(\mathbf{y} - \mathbf{X}\hat{\beta}) \\ &= \mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\hat{\beta} \\ &= \mathbf{X}^T\mathbf{y} - (\mathbf{X}^T\mathbf{X})(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\,\mathbf{y} = \mathbf{0} \end{aligned}$$

- The cross-product between the fitted value $\hat{y}$ and the residual vector r is zero, i.e.
$$\hat{\mathbf{y}}^T \mathbf{r} = \hat{\beta}^T \mathbf{X}^T \mathbf{r} = \mathbf{0}$$
This implies that the residual vector $\mathbf{r}$ is **orthogonal to each column of X and to $\hat{\mathbf{y}}$**.

2.3 Least-Squares & Normal Equations

In this section, we focus on the system of linear equations that we solve to obtain the least squares estimators for β. Recall, that we want to find a a vector $\widehat{\beta}$ that minimizes:
$$\min_{\beta} ||\mathbf{y} - \mathbf{X}\beta||^2$$
Any vector that provides a minimum value for this expression is called a least-squares solution.

The set of all least squares solutions is precisely the **set of solutions** to
$$(\mathbf{X}^T\mathbf{X})\beta = \mathbf{X}^T\mathbf{y}$$
There is a *unique* solution **if and only if** $rank(\mathbf{X}) = p+1$ in which case $(\mathbf{X}^T\mathbf{X})$ is invertible.

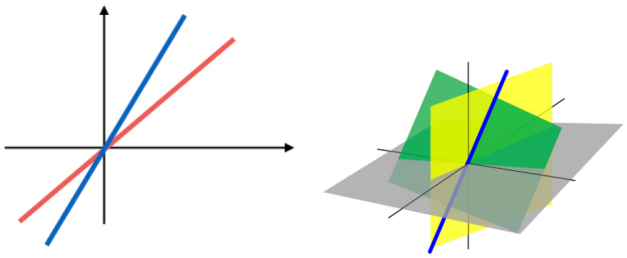
System of Linear Equations

Linear Algebra Review Let us review a few facts from linear algebra related to solutions to a system of linear equations. For simplicity in the notation, we use the *generic* system of equations:
$$\mathbf{A}z = c$$
where
$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mk} \end{pmatrix}, \quad z = \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{pmatrix}, \quad c = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{pmatrix}$$
Denote by $\mathbf{A}_i = (a_{1i}, a_{2i}, ..., a_{mi})^T$ the vector representing the *columns* of $\mathbf{A}$. Therefore, $\mathcal{C}(\mathbf{A})$ is the space generated by the columns of $\mathbf{A}$, or in other words the $span(\mathbf{A}_1, \mathbf{A}_2, ..., \mathbf{A}_k)$.

Definition

The **span** of a collection of vectors $(\mathbf{A}_1, \mathbf{A}_2, ..., \mathbf{A}_k)$ is the set of all linear combinations of these vectors:
$$span(\mathbf{A}_1, \mathbf{A}_2, ..., \mathbf{A}_k) = \Big\{ d_1\mathbf{A}_1 + d_2\mathbf{A}_2 + ... + d_k\mathbf{A}_k, \text{ for any constants } d_1, ..., d_k \in \mathbb{R} \Big\}$$

The column space of $\mathbf{A}$, $\mathcal{C}(\mathbf{A})$, is a **subspace** in $\mathbb{R}^n$. A linear subspace of $\mathbb{R}^n$ is thought of as a "flat" surface within $\mathbb{R}^n$, and it is a collection of vectors that is closed under linear combinations. Illustration of subspaces of $\mathbb{R}^n$ are shown below:



Solving the System of Equations $\mathbf{A}z = c$

For a system of equations, $\mathbf{A}z = c$, to have a solution, c must be a linear combination of the columns of $\mathbf{A}$, i.e. $c \in \mathcal{C}(\mathbf{A})$. This is simply obtained by the definition of *matrix multiplication* and *equality*:

$$\mathbf{A}z = c \Leftrightarrow c = z_1\mathbf{A}_1 + ... + z_k\mathbf{A}_k$$

When $c \notin \mathcal{C}(\mathbf{A})$, we need to **find $\hat{c}$ living in $\mathcal{C}(\mathbf{A})$ that is closest to** c. If this is the case, $\mathbf{A}z = \hat{c}$ has a unique solution, and $\hat{c}$ comes as close to the original data as possible. To do so, we need to **project** c **orthogonally** onto $\mathcal{C}(\mathbf{A})$, by multiplying both sides by $\mathbf{A}^T$:

$$\mathbf{A}^T\mathbf{A}z = \mathbf{A}^Tc$$

* These are the normal equations.

In the sketch to the right, $\mathcal{C}(\mathbf{A})$, the space spanned by the columns of $\mathbf{A}$, is a *flat* surface, and c is a point that exists off of that flat surface. The **shortest distance** from the point c to the plane $\mathcal{C}(\mathbf{A})$ is the one *orthogonal* to the plane.

The *normal equations* essentially help us find the closest point to c that belongs $\mathcal{C}(\mathbf{A})$ by means of an orthogonal projection.

Geometric Representation of LS

Taking the previous discussion to the linear regression framework:

- $\mathcal{C}(\mathbf{X})$ is the space that is *spanned* by the predictors X_1, X_2, ..., X_p, or in other words the columns of the design matrix. It is a *flat* subspace of $\mathbb{R}^n$.
- The response $\mathbf{y}$ is a vector **off** the subspace, and the fitted value $\hat{\mathbf{y}}$ is the orthogonal projection of $\mathbf{y}$ onto $\mathcal{C}(\mathbf{X})$.

- The residuals, on the other hand, $\mathbf{r} = \mathbf{y} - \hat{\mathbf{y}}$ are **orthogonal** to $\hat{\mathbf{y}}$ and to $\mathcal{C}(\mathbf{X})$.

Essentially, what the least-squares method does is that it decomposes the data vector $\mathbf{y}$ into two orthogonal components:

$$\mathbf{y}_{n\times 1} = \hat{\mathbf{y}}_{n\times 1} + \mathbf{r}_{n\times 1}$$

2.4 Goodness-Of-Fit: R-Square

A measure of how well the model fits the data is the **R-square** or the so-called **coefficient of determination** or *percentage of variance explained*. It is defined as

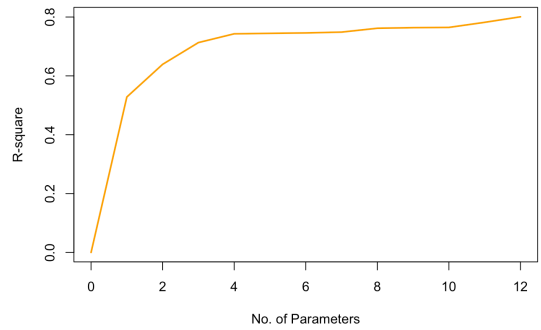
$$\begin{aligned} R^2 = \frac{\sum_i (\hat{y}_i - \bar{y})^2}{\sum_i (y_i - \bar{y})^2} &= \frac{\text{distance of model from grand mean}}{\text{distance of observations from grand mean}} \\ &= \frac{||\hat{\mathbf{y}} - \bar{\mathbf{y}}||^2}{||\mathbf{y} - \bar{\mathbf{y}}||^2} \\ &= \frac{||\mathbf{y} - \bar{\mathbf{y}}||^2 - ||\hat{\mathbf{y}} - \mathbf{y}||^2}{||\mathbf{y} - \bar{\mathbf{y}}||^2} \\ &= 1 - \frac{||\hat{\mathbf{y}} - \mathbf{y}||^2}{||\mathbf{y} - \bar{\mathbf{y}}||^2} := 1 - \frac{RSS}{TSS} \end{aligned}$$

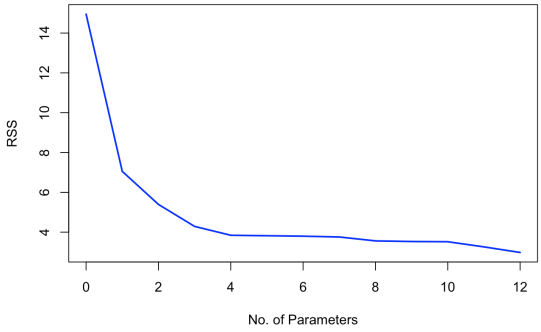
where we used orthogonality to get

$$\underbrace{||\mathbf{y} - \bar{\mathbf{y}}||^2}_{\substack{\text{total} \\ \text{variation}}} = \underbrace{||\hat{\mathbf{y}} - \bar{\mathbf{y}}||^2}_{\substack{\text{model variation} \\ \text{from mean}}} + \underbrace{||\hat{\mathbf{y}} - \mathbf{y}||^2}_{\text{error}}$$

Properties of R^2

- $0 \leq R^2 \leq 1$
- R^2 is invariant of any location and/or scale change of Y or X.
- R^2 alone does not tell us much about the effectiveness of the LS method.
- A small R^2 does not imply that the LS model is bad.
- Adding a new predictor, even if it is randomly generated and has nothing to do with Y will decrease RSS and therefore increase R^2. These relationships are illustrated in the two plots below:





2.5 Linear Transformations on X

Suppose we have a linear regression model of Y on X. If we **scale** or **shift** a predictor, then the fitted values, $\hat{\mathbf{y}}$, and R^2 stay the same, but LS estimators, $\hat{\beta}$, change!

Some Examples

- As an example, consider scaling a predictor, i.e. $\tilde{x}_{i2} = 2x_{i2}$ or $\tilde{x}_{i3} = x_{i3}/4$. This transformation will not impact $\hat{\mathbf{y}}$, or R^2.

- In the `birthweight` example, the length of a baby is given in cm. If we want to change the unit to inches by dividing by 2.54 the column that corresponds to length, i.e. $\tilde{X}_1 = X_1/2.54$, then this won't affect our predicted values.

These statements hold true, if we apply any linear transformation on the p predictors **as long as the transformation does not change the rank of X**.

2.6 Rank deficiency

The design matrix $\mathbf{X}$ is an $n \times p$ matrix. If this matrix is **not of full rank** (i.e., its columns are not linearly independent), the matrix $\mathbf{X}^T\mathbf{X}$ can not be inverted. This implies that there is some redundancy in the columns of $\mathbf{X}$, i.e. one column can be written as a linear combination of other columns. As a result, the matrix $\mathbf{X^T X}$ is *singular* which means that the LS solution is not unique (*identifiability problem*).

However, $\mathcal{C}(\mathbf{X})$ is well defined and thus $\hat{y}$ is well-defined and can be computed. `R`/ `Python` can cope well with this problem, and the returned model can be used for prediction.

2.7 Hypothesis Testing in MLR

When fitting a regression model, we often want to understand which predictors are statistically significant or which model (a larger or a smaller) is more appropriate –either for prediction or for estimation purposes–.

Testing for the statistical significance of one (or more) predictor(s), can be formulated as testing whether the corresponding β is zero, or as a model comparison test.In this class, we summarize all testing questions as the following hypothesis test:

$$\begin{cases} H_0: \text{ Reduced Model with } p_0 \text{ coefficients} \\ H_\alpha: \text{ Full/ Larger Model with } p_\alpha \text{ coefficients} \end{cases}$$

Obviously, $p_\alpha > p_0$, since the reduced model is a subset/special case of the full model. Therefore, we expect the reduced model to have RSS_0 smaller than RSS_α, the full model's RSS, since $p_\alpha > p_0$. In this context, the main tool for testing is the so-called **partial F test** and is formulated as follows:

Partial F test

Hypothesis Test:

$$\begin{cases} H_0: \text{ Reduced Model with } p_0 \text{ coefficients} \\ H_\alpha: \text{ Full/ Larger Model with } p_\alpha \text{ coefficients} \end{cases}$$

Test Statistic:

$$F = \frac{(RSS_0 - RSS_\alpha)/p_\alpha - p_0}{RSS_\alpha/(n - p_\alpha)} \sim F_{p_\alpha - p_0, n - p_\alpha}$$

Decision Rule: Reject H_0, if the F test statistic is large (or larger than $F_{p_\alpha - p_0, n - p_\alpha}$), i.e. the variation missed by the reduced model, when being compared with the error variance, is significantly large.

- The **numerator** in the partial F test quantifies the variation in the data not explained by the reduced model, but explained by the full model.
- On the other hand, the **denomimator** is equal to the variation in the data not explained by the full model (i.e., not explained by either model), which is used to estimate the error variance.

Birthweight Example (cont'd)

We want to decide whether the group of the predictors that refer to father's characteristics (i.e. variables X_9 – X_{12}) are significant –as a group.

The hypothesis test is formulated as

$$\begin{cases} H_0 : Y \sim 1 + \beta_1 X_1 + ... + \beta_8 X_8 \\ H_\alpha : Y \sim 1 + \beta_1 X_1 + ... + \beta_8 X_8 + ... + \beta_{12} X_{12} \end{cases}$$

In this case $p_0 = 8+1$ is the number of β's in the reduced model, and $p_\alpha = 12+1$ is the number of β's in the full model. Using `R`/`Python`, we extract RSS_0, RSS_α to perform the partial F test and make a decision.

It turns out that $F = 0.7225$ is smaller than $F_{p_\alpha - p_0, n-p_\alpha} = 2.7$ meaning that the model under the null (reduced) is preferred.

Partial F Test: Special Cases

- **Test for a Single Predictor β_j**

$$\begin{cases} H_0 : Y \sim 1 + \beta_1 X_1 + ... + \beta_{j-1} X_{j-1} + \qquad \beta_{j+1} X_{j+1} + ... + \beta_p X_p \\ H_\alpha : Y \sim 1 + \beta_1 X_1 + ... + \beta_{j-1} X_{j-1} + \beta_j X_j + \beta_{j+1} X_{j+1} + ... + \beta_p X_p \end{cases}$$

is equivalent to the so-called **t-test for each regression parameter**.

- **Test for all Predictors**

$$\begin{cases} H_0 : Y \sim 1 \qquad \text{[intercept-only model]} \\ H_\alpha : Y \sim 1 + \beta_1 X_1 + ... + \beta_{j-1} X_{j-1} + \beta_j X_j + \beta_{j+1} X_{j+1} + ... + \beta_p X_p \end{cases}$$

This is the **overall** F test that can be thought of as a *goodness-of-fit test*.

2.8 Categorical Variables in MLR

As we discussed some of the predictors may be *categorical*. Assume a predictor that is categorical with k levels. When added in a linear regression model, it is coded as $k-1$ *numerical predictors*. The default coding is of the form:

$$D_i = \begin{cases} 0, & \text{if not level } i \\ 1, & \text{if level } i \end{cases}$$

where Level 1 is the *reference level*.

An Example with Categorical Variables

Consider a categorical predictor with three levels (a, b, c):

$$\begin{pmatrix} a \\ a \\ b \\ b \\ b \\ c \\ c \end{pmatrix} \Longrightarrow \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

In this setup,

- Col 1 corresponds to level a, i.e. $X_1 = 1$ if in column 1, and 0 otherwise.
- Col 2 corresponds to level b, i.e. $X_2 = 1$ if in column 2, and 0 otherwise.
- Col 3 corresponds to level c: this is the reference level, since its value is absorbed by the intercept.

  In general, any level can be chosen to be the reference level; this is more a choice that depends on the context rather than the statistics behind. There are also alternative ways to code the variables. If we choose a different coding, then this will affect the estimated regression coefficients (and the interpretation), but it won't affect the predicted/fitted values.

2.9 Collinearity

In practice, we often encounter problems in which many of the predictors are **highly correlated** with each other. In such cases, the fitted values and sampling variance of the regression coefficients can be highly dependent on the particular predictors chosen for the model. This results in sensitive and unreliable estimators.

Possible *symptoms* of collinearity include :

(i) high pair-wise (sample) correlation between predictors,

(ii) relatively large R^2,

(iii) statistically significant F test without *any* statistically significant predictors.

There are also specific tests and metrics that allow us to assess the severity of collinearity in our model, but this is beyond the scope of this course.

The easiest way to **remedy** collinearity is to remove some predictors from highly correlated groups of predictors. Other (more advanced) approaches include PCA, or regularization using penalized Least Squares.

2.10 Model Diagnostics

We made several assumptions when defining the linear regression model:

$$\mathbf{y} = \mathbf{X}\beta + \varepsilon, \quad \text{where} \quad \varepsilon \sim IID(0, \sigma^2)$$

This means that we assumed:

- *Linearity*, i.e. the model is linear with respect to the predictors.
- *Constant Variance*, i.e. all Y's have common, constant variance.
- *Uncorrelated errors*, i.e. Y_i and Y_j are uncorrelated for $i \neq j$.
- *(Normality)*: this is not part of the assumptions above, and it is not required for the derivation of the least-squares estimators. However, it is needed when one wants to discuss statistical properties of the estimators and the fitted values and residuals. It is, in particular, one of the most critical assumptions when it comes to hypothesis testing.

If we are interested in using the model for understanding the effect of the predictors on the response (i.e. not purely for prediction purposes), then we need to make sure that all assumptions are satisfied (via diagnostic test/plots), and if not take appropriate actions (remedial measures).

Since the focus of this class is prediction, we refer the students to a regression course or textbook to review diagnostics and remedial measures for a MLR model. As an example, you can refer to …

Outliers

Finally, outliers might be a concern both in large and smaller data sets. If the data set is not very large, it is suggested to test for outliers, e.g. using outlier tests based on the leave-one-out prediction error. Otherwise, if the data set is really large, then this test is not useful and it is suggested to adjust for multiple comparisons or consider alternative approaches.

If outliers seem to be an issue, then it is important to (i) know the range of each variable, (ii) apply log, square-root or other transformations on right skewed predictors and Y, or (iii) apply winsorization to remove the effect of extreme values.

2.11 The `Birthweight` Data Set Example

The `birthweight.csv` data set contains data from a study on the birth weight of **42** babies. The variables in the data set are the following:

- `ID`: Unique Identification number of a baby
- `Length`: Length of the baby at time of birth in cm (X_1)
- `Birthweight`: Weight of the baby at time of birth in kg (Y)
- `Headcirc`: Head circumference of the baby at time of birth in cm (X_2)
- `Gestation`: Completed weeks of gestation (X_3)
- `smoker`: Mother is/is not a smoker {0: No, 1: Yes} (X_4)
- `mage`: Mother's age at time of birth (X_5)
- `mnocig`: Mother's number of cigarettes smoked per day (X_6)
- `mheight`: Mother's height in cm (X_7)
- `mppwt`: Mother's pre-pregnancy weight (X_8)
- `fage`: Father's age at time of birth (X_9)
- `fedyrs`: Father's years of education (X_{10})
- `fnocig`: Father's number of cigarettes smoked per day (X_{11})
- `fheight`: Father's height (X_{12})

```
birthweight <- read.csv("data/week2/Birthweight.csv", header=TRUE)
dim(birthweight)
```

```
## [1] 42 14
```

```
head(birthweight)
```

```
##     ID Length Birthweight Headcirc Gestation smoker mage mnocig mheight mppwt
## 1 1360     56        4.55       34        44      0   20      0     162    57
## 2 1016     53        4.32       36        40      0   19      0     171    62
## 3  462     58        4.10       39        41      0   35      0     172    58
## 4 1187     53        4.07       38        44      0   20      0     174    68
## 5  553     54        3.94       37        42      0   24      0     175    66
## 6 1636     51        3.93       38        38      0   29      0     165    61
##   fage fedyrs fnocig fheight
## 1   23     10     35     179
## 2   19     12      0     183
## 3   31     16     25     185
## 4   26     14     25     189
## 5   30     12      0     184
## 6   31     16      0     180
```

Since the `ID` variable is not of interest in our analysis, we won't use it in our analysis. For convenience can remove it from the data frame we are working with:

```
birthweight = birthweight[,-1]
```

2.11.0.1 Fitting a Multiple Linear Regression

We start by fitting a linear regression model using all the predictors and we print the summary of the LS results:

```
birthweight.mlr.full = lm(Birthweight~., data=birthweight)
summary(birthweight.mlr.full)
```

```
##
## Call:
## lm(formula = Birthweight ~ ., data = birthweight)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.37925 -0.25558 -0.05541  0.21700  0.60329
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -3.812311   2.080150  -1.833  0.07713 .
## Length       0.034152   0.031456   1.086  0.28656
## Headcirc     0.086594   0.029489   2.937  0.00644 **
## Gestation    0.096565   0.031323   3.083  0.00447 **
## smoker      -0.243895   0.168516  -1.447  0.15854
## mage        -0.019004   0.018537  -1.025  0.31374
## mnocig       0.001064   0.006854   0.155  0.87770
## mheight      0.004187   0.014153   0.296  0.76944
## mppwt        0.010346   0.011228   0.921  0.36443
## fage         0.007187   0.016327   0.440  0.66305
## fedyrs       0.003220   0.030939   0.104  0.91783
## fnocig       0.003445   0.004013   0.858  0.39769
## fheight     -0.013293   0.009614  -1.383  0.17731
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3342 on 29 degrees of freedom
## Multiple R-squared:  0.7834, Adjusted R-squared:  0.6938
## F-statistic: 8.742 on 12 and 29 DF,  p-value: 1.011e-06
```

If we want to check the objects returned by `lm` and `summary(lm...)`:

```
names(birthweight.mlr.full)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

```
names(summary(birthweight.mlr.full))
```

```
##  [1] "call"          "terms"         "residuals"     "coefficients"
```

```
##  [5] "aliased"       "sigma"         "df"            "r.squared"
##  [9] "adj.r.squared" "fstatistic"    "cov.unscaled"
```

The design matrix in this example can be extracted using:

```
design.birthweight = model.matrix(birthweight.mlr.full);
```

If we want to extract the Regression coefficients including the intercept, i.e. $\hat{\beta}$, we have

```
birthweight.mlr.full$coef
```

```
##  (Intercept)       Length     Headcirc    Gestation       smoker         mage
## -3.812311183  0.034151922  0.086594329  0.096565435 -0.243895136 -0.019003806
##       mnocig      mheight        mppwt         fage       fedyrs       fnocig
##  0.001064074  0.004187315  0.010346096  0.007187428  0.003219952  0.003444674
##      fheight
## -0.013293205
```

The fitted values and residuals are obtained by:

```
full.model.residuals = birthweight.mlr.full$resid
full.model.fitted = birthweight.mlr.full$fitted
```

Check the calculation for Residual standard error and Multiple R-squared in the summary output:

The residuals **r** are used to estimate the *error variance*:

$$\hat{\sigma}^2 = \frac{1}{n-p-1}\sum_i r_i^2 = \frac{RSS}{n-p-1}$$

To obtain and estimator for $\hat{\sigma}$, we can either use the formula (from the notes):

```
n = dim(birthweight)[1]
p = dim(birthweight)[2] - 1
sqrt(sum(birthweight.mlr.full$residuals^2)/(n- p - 1))
```

```
## [1] 0.3341631
```

or we can extract it from the `lm` object:

```
summary(birthweight.mlr.full)$sigma
```

```
## [1] 0.3341631
```

The coefficient of determination computes as:

$$\frac{\sum_i(\hat{y}_i-\bar{y})^2}{\sum_i(y_i-\bar{y})^2} = 1 - \frac{RSS}{TSS}$$

We can extract R^2 from the summary output or compute it:

```
summary(birthweight.mlr.full)$r.squared
```

```
## [1] 0.783425
```

```
1 - sum(birthweight.mlr.full$residuals^2)/(var(birthweight$Birthweight)*(n-1))
```

```
## [1] 0.783425
```

```
1 - var(birthweight.mlr.full$residuals)/var(birthweight$Birthweight)
```

```
## [1] 0.783425
```

2.11.0.2 Invariance of R^2 & Fitted Values

We can also check the invariance property of R^2 by rescaling one of the variables:

```
## Center the Length variable
birthweight.new = birthweight
birthweight.new$Length.new = birthweight.new$Length - rep(mean(birthweight.new$Length), 42)

## Model with original `Length`
summary(lm(Birthweight~., data=birthweight.new[-14]))$r.squared
```

```
## [1] 0.783425
```

```
## Model with centered `Length`
summary(lm(Birthweight~., data=birthweight.new[-1]))$r.squared
```

```
## [1] 0.783425
```

If we want, we can also check that the fitted values from both models are identical:

```
## Difference in fitted values between Model w `Length` -  Model w `centered-Length`
### The values are added to make the output cleaner.

sum( fitted(lm(Birthweight~., data=birthweight.new[-14])) - fitted(lm(Birthweight~., data=birthwe
```

```
## [1] -8.881784e-16
```

Note that the difference is not exactly 0, since there are rounding errors that accumulated when the various R functions are applied.

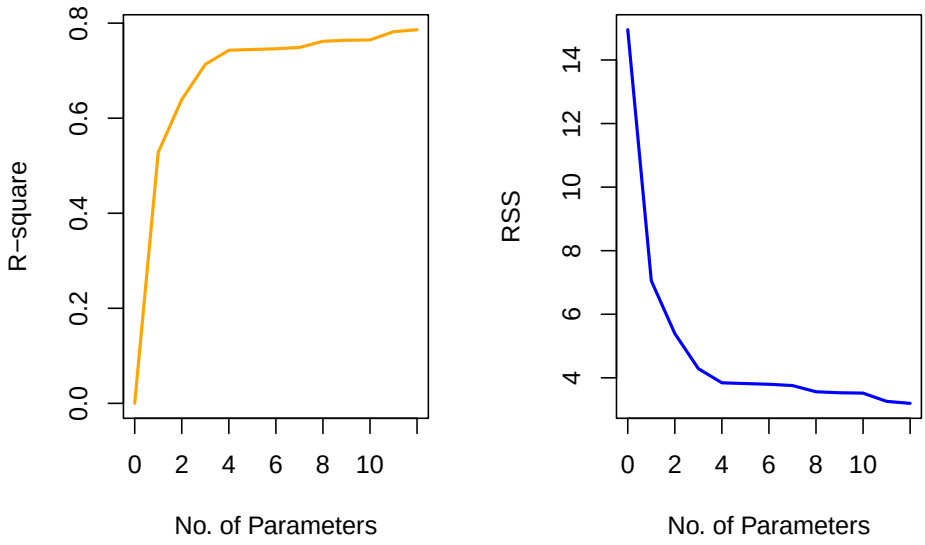
2.11.0.3 R^2 and RSS behavior when adding more Xs

We can also visualize how the R^2 and RSS change as a function of the number of variables in the model to observe that adding a new predictor, even if it is randomly generated and has nothing to do with Y will decrease RSS and therefore increase R^2.

So, start with the null model that contains only the intercept and then we add variables one at a time, keeping the R^2 and RSS that we will plot in the end. We will also randomly generate a new variable `randomvar` to illustrate that even an unrelevant variable has an effect on both R^2 and RSS:

```
birthweight.new = birthweight
birthweight.new$randomvar = rnorm(42)
```

```
no.var = 0:12
par(mfrow=c(1,2))
plot(no.var, rsq, xlab="No. of Parameters", ylab = "R-square", type='n');
lines(no.var, rsq, lwd=2, col="orange")
plot(no.var, rss, xlab="No. of Parameters", ylab = "RSS", type='n');
lines(no.var, rss, lwd=2, col="blue")
```



This is to illustrate why R^2 and RSS are not good metrics when it comes to model selection. They are useful, because they helps us understand the quality of the model or RSS is a key ingredient in constructing hypothesis tests, but they are never used for model selection.

2.11.0.4 Rank Deficiency

We have rank deficiency when $\mathbf{X}$ is not of full rank which leads to the $(\mathbf{X}^T\mathbf{X})$ matrix not being invertible.

Here we illustrate a scenario where it is easy to identify the cause of the problem.

```
## Create a new variable: the average height of both parents
birthweight.new = birthweight
birthweight.new$aveheight = (birthweight.new$mheight+birthweight.new$fheight)/2
```

```
## Regress birthweight against all predictors
summary(lm(Birthweight ~ ., data=birthweight.new))
```

```
##
## Call:
## lm(formula = Birthweight ~ ., data = birthweight.new)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.37925 -0.25558 -0.05541  0.21700  0.60329
##
## Coefficients: (1 not defined because of singularities)
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -3.812311   2.080150  -1.833  0.07713 .
## Length       0.034152   0.031456   1.086  0.28656
## Headcirc     0.086594   0.029489   2.937  0.00644 **
## Gestation    0.096565   0.031323   3.083  0.00447 **
## smoker      -0.243895   0.168516  -1.447  0.15854
## mage        -0.019004   0.018537  -1.025  0.31374
## mnocig       0.001064   0.006854   0.155  0.87770
## mheight      0.004187   0.014153   0.296  0.76944
## mppwt        0.010346   0.011228   0.921  0.36443
## fage         0.007187   0.016327   0.440  0.66305
## fedyrs       0.003220   0.030939   0.104  0.91783
## fnocig       0.003445   0.004013   0.858  0.39769
## fheight     -0.013293   0.009614  -1.383  0.17731
## aveheight          NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3342 on 29 degrees of freedom
## Multiple R-squared:  0.7834, Adjusted R-squared:  0.6938
## F-statistic: 8.742 on 12 and 29 DF,  p-value: 1.011e-06
```

As you see the β_j that corresponds to the new variable cannot be calculated (hence the `NA`), since `aveheight` is linearly dependent on both `mheight` and `fheight`. At the same time, although the matrix $\mathbf{X}$ is not of full rank, `R` produces solutions for the rest of the βs and the fitted model is valid for prediction purposes. The predictions derived from this model will align with those obtained from fitting a linear regression model excluding the columns with NA coefficient.

This issue can be easily fixed, by removing one or more variables. Which variable will be removed depends on the context. For example, we need to (conceptually) answer the question on whether we prefer to have the `aveheight` or each parent's height individually.

A harder issue to resolve is when two predictors are not exaclty linearly depen-

dent, but approximately - this means that they are highly correlated, but not an exact function of one another. This is more challenging to deal with and we will discuss it later.

2.11.0.5 Test for the Significance of Predictors

We use the partial F test

$$F = \frac{(RSS_0 - RSS_\alpha)/p_\alpha - p_0}{RSS_\alpha/(n - p_\alpha)} \sim F_{p_\alpha - p_0, n - p_\alpha}$$

to compare the full model with all available predictors versus the model that exclude father-related variables. That is, the hypothesis test is formulated as

$$\begin{cases} H_0 : Y \sim 1 + \beta_1 X_1 + ... + \beta_8 X_8 \\ H_\alpha : Y \sim 1 + \beta_1 X_1 + ... + \beta_8 X_8 + ... + \beta_{12} X_{12} \end{cases}$$

```
## Full Model was fitted before
## and is stored in birthweight.mlr.full

## Reduced Model
birthweight.mlr.reduced1 = lm(Birthweight~ Length+Headcirc+Gestation+smoker+mage+mnocig
```

We can use one function to perform the test:

```
anova(birthweight.mlr.reduced1, birthweight.mlr.full)
```

```
## Analysis of Variance Table
##
## Model 1: Birthweight ~ Length + Headcirc + Gestation + smoker + mage +
##     mnocig + mheight + mppwt
## Model 2: Birthweight ~ Length + Headcirc + Gestation + smoker + mage +
##     mnocig + mheight + mppwt + fage + fedyrs + fnocig + fheight
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1     33 3.5610
## 2     29 3.2383  4   0.32271 0.7225 0.5837
```

The F statistic value is 0.7225, and the corresponding p-value is 0.5837 which means that we fail to reject the null and conclude that the model under the H_0 is preferred. In other words, we prefer the smaller model which implies that the father-related predictors can be removed from the model.

We can also build the F statistic, by extracting all necessary components:

```
RSS0 = deviance(birthweight.mlr.reduced1)
RSS0
```

```
## [1] 3.560996
```

```
RSSalpha = deviance(birthweight.mlr.full)
RSSalpha
```

```
## [1] 3.238285
```

```
p0 =9;
palpha = 13;
n=42;

## Partial F test
F = ((RSS0 - RSSalpha)/(palpha-p0))/(RSSalpha/(n-palpha))
F
```

```
## [1] 0.7224983
```

```
## Critical value use to compare with the F statistic.
qf(0.95, palpha-p0, n-palpha)
```

```
## [1] 2.701399
```

the conclusion is the same since $F < F_{p_\alpha - p_0, n-p_\alpha}$.

2.11.0.6 Testing for a Single β_j

Let's test for the significance of variable `Length` (X_1), i.e.

$$\begin{cases} H_0 : Y \sim 1 + \qquad\quad \beta_2 X_2 + ... + \beta_{12} X_{12} \\ H_\alpha : Y \sim 1 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_{12} X_{12} \end{cases}$$

We fit the full model and then compare by means of the partial F test:

```
## Full Model was fitted before
## and is stored in birthweight.mlr.full

## Reduced Model
birthweight.mlr.reduced2 = lm(Birthweight~ Headcirc+Gestation+smoker+mage+mnocig+mheight+mppwt+fa

## Partial $F$ test
anova(birthweight.mlr.reduced2, birthweight.mlr.full)
```

```
## Analysis of Variance Table
##
## Model 1: Birthweight ~ Headcirc + Gestation + smoker + mage + mnocig +
##     mheight + mppwt + fage + fedyrs + fnocig + fheight
## Model 2: Birthweight ~ Length + Headcirc + Gestation + smoker + mage +
##     mnocig + mheight + mppwt + fage + fedyrs + fnocig + fheight
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1     30 3.3699
## 2     29 3.2383  1   0.13162 1.1787 0.2866
```

```
## Compare the $F$ value above with the square of the $t$ test
## for the length variable in the output below:
summary(birthweight.mlr.full)
```

```
##
## Call:
## lm(formula = Birthweight ~ ., data = birthweight)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.37925 -0.25558 -0.05541  0.21700  0.60329
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -3.812311   2.080150  -1.833  0.07713 .
## Length       0.034152   0.031456   1.086  0.28656
## Headcirc     0.086594   0.029489   2.937  0.00644 **
## Gestation    0.096565   0.031323   3.083  0.00447 **
## smoker      -0.243895   0.168516  -1.447  0.15854
## mage        -0.019004   0.018537  -1.025  0.31374
## mnocig       0.001064   0.006854   0.155  0.87770
## mheight      0.004187   0.014153   0.296  0.76944
## mppwt        0.010346   0.011228   0.921  0.36443
## fage         0.007187   0.016327   0.440  0.66305
## fedyrs       0.003220   0.030939   0.104  0.91783
## fnocig       0.003445   0.004013   0.858  0.39769
## fheight     -0.013293   0.009614  -1.383  0.17731
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3342 on 29 degrees of freedom
## Multiple R-squared:  0.7834, Adjusted R-squared:  0.6938
## F-statistic: 8.742 on 12 and 29 DF,  p-value: 1.011e-06
```

```
## The square of the t-value is obtained by
summary(birthweight.mlr.full)$coefficients[2,3]^2
```

```
## [1] 1.178721
```

2.11.0.7 Testing for all the predictors

The F test statistic and corresponding p-value in the `summary` output correspond to the following hypothesis:

$$\begin{cases} H_0 : Y \sim 1 \\ H_\alpha : Y \sim 1 + \beta_1 X_1 + ... + \beta_{j-1} X_{j-1} + \beta_j X_j + \beta_{j+1} X_{j+1} + ... + \beta_p X_p \end{cases}$$

which is the so-called **overall F test**.

```
summary(birthweight.mlr.full)$fstatistic
```

```
##     value     numdf     dendf
##  8.741902 12.000000 29.000000
```

In this case, the F value is equal to 8.74 and it follows an $F_{12,29}$ distribution.The corresponding p-value is equal to 1.011e-06 which is much less than $\alpha = 0.05$. This implies that we reject the null and conclude that the model with all the predictors is more adequate compared to the intercept-only model.

2.11.0.8 Categorical Variables

In the `birthweight` data set we only have binary variables, so for illustration purposes, we will create a new categorical variable, `mage.group` with 3 levels: a=below 20, b=20-30, and c=above 30:

```
birthweight.new = birthweight
birthweight.new$mage.group <- as.factor(ifelse(birthweight.new$mage <20, 'a', ifelse(birthweight.
```

Let's fit the model with the age group variable and look at the design matrix:

```
birthweight.mlr.categorical = lm(Birthweight~., data=birthweight.new[,-6])
model.matrix(birthweight.mlr.categorical)
```

```
##    (Intercept) Length Headcirc Gestation smoker mnocig mheight mppwt fage
## 1            1     56       34        44      0      0     162    57   23
## 2            1     53       36        40      0      0     171    62   19
## 3            1     58       39        41      0      0     172    58   31
## 4            1     53       38        44      0      0     174    68   26
## 5            1     54       37        42      0      0     175    66   30
## 6            1     51       38        38      0      0     165    61   31
## 7            1     52       34        40      0      0     157    50   31
## 8            1     53       33        42      0      0     165    61   21
## 9            1     54       38        38      0      0     172    50   20
## 10           1     50       35        38      0      0     157    48   22
## 11           1     53       33        41      0      0     164    62   37
## 12           1     51       36        40      0      0     168    53   29
## 13           1     52       36        38      0      0     164    57   35
## 14           1     53       33        41      0      0     155    55   25
## 15           1     53       34        40      0      0     167    60   30
## 16           1     48       33        37      0      0     158    54   39
## 17           1     48       35        39      0      0     162    62   27
## 18           1     48       33        34      0      0     167    64   25
## 19           1     53       34        39      0      0     165    57   23
## 20           1     43       32        33      0      0     149    45   26
## 21           1     53       38        40      1      2     170    59   24
## 22           1     51       33        41      1      7     160    53   24
```

```
## 23           1     50       30       37       1       7     165     60     20
## 24           1     50       35       39       1       7     159     52     23
## 25           1     48       30       37       1       7     163     47     20
## 26           1     46       32       35       1       7     166     57     37
## 27           1     48       30       33       1       7     161     50     20
## 28           1     58       39       41       1      12     173     70     38
## 29           1     53       34       40       1      12     163     49     41
## 30           1     49       36       40       1      12     152     48     37
## 31           1     52       35       38       1      12     165     64     38
## 32           1     51       38       39       1      17     157     48     32
## 33           1     50       33       39       1      17     156     53     24
## 34           1     50       33       45       1      25     163     54     30
## 35           1     52       36       39       1      25     170     78     40
## 36           1     53       37       41       1      25     161     66     46
## 37           1     52       37       40       1      25     170     62     30
## 38           1     52       33       39       1      25     181     69     23
## 39           1     49       34       38       1      25     162     57     32
## 40           1     53       34       41       1      35     163     51     31
## 41           1     47       33       35       1      35     170     57     23
## 42           1     53       32       40       1      50     168     61     31
##    fedyrs fnocig fheight mage.groupb mage.groupc
## 1      10     35     179           1           0
## 2      12      0     183           0           0
## 3      16     25     185           0           1
## 4      14     25     189           1           0
## 5      12      0     184           1           0
## 6      16      0     180           1           0
## 7      16      0     173           1           0
## 8      10     25     185           1           0
## 9      12      7     172           0           0
## 10     14      0     179           1           0
## 11     14      0     170           1           0
## 12     16      0     181           1           0
## 13     16      0     183           0           1
## 14     14     25     183           1           0
## 15     16     25     182           1           0
## 16     10      0     171           1           0
## 17     14      0     178           1           0
## 18     12     25     175           1           0
## 19     14      2     193           0           0
## 20     16      0     169           1           0
## 21     12     12     185           1           0
## 22     16     12     176           1           0
## 23     14      0     183           0           0
## 24     14     25     200           1           0
## 25     10     35     185           1           0
```

```
## 26     14     25     173            0            1
## 27     10     35     180            1            0
## 28     14     25     180            0            1
## 29     12     50     191            0            1
## 30     12     25     170            1            0
## 31     14     50     180            1            0
## 32     12     25     169            1            0
## 33     12      7     179            1            0
## 34     16      0     183            1            0
## 35     16     50     178            1            0
## 36     16      0     175            0            1
## 37     10     25     181            1            0
## 38     16      2     181            1            0
## 39     16     50     194            0            1
## 40     16     25     185            1            0
## 41     12     50     186            1            0
## 42     16      0     173            0            1
## attr(,"assign")
##  [1]  0  1  2  3  4  5  6  7  8  9 10 11 12 12
## attr(,"contrasts")
## attr(,"contrasts")$mage.group
## [1] "contr.treatment"
```

When fitting the regression model, R automatically converted the factor to the appropriate number of indicator/variables which were included in the design matrix. We observe that level a is the baseline and the two generated columns are levels b and c.

In general, the reference level can be changed as well as the coding. Those changes will affect the estimated β coefficients, but they will not affect the fitted or predicted values.

2.11.0.9 Mean Response Estimation & Prediction

In practice, we have target values for which we need to estimate the mean response of values for which we want to do prediction.

Since this is not the case here, in order to illustrate both methods, we use as inputs the mean value of every predictor. This is done as follows:

```
meanvalue = apply(birthweight[,-2], 2, mean)
```

Now, we create the X `data.frame` to use as an *input* in the `predict()` function:

```
x=data.frame(t(meanvalue))
```

Using the predict function we have:

```
predict.lm(birthweight.mlr.full, x)
```

```
##        1
## 3.312857
```

2.11.0.10 Training vs. Testing Error

In this example, and in linear regression models, as we introduce more variables, training error (measured by RSS or MSE) consistently decreases. However, this reduction in training error does not guarantee a corresponding decrease in test error, which measures prediction accuracy on independent test data. To demonstrate this, we randomly split our dataset into 60% training and 40% test portions, progressively adding predictors. This approach highlights how additional predictors can lower training error while test error may not follow the same trend!

```
n = dim(birthweight)[1]      ## sample size
p = dim(birthweight)[2]-1    ## number of non-intercept predictors

ntrain =  round(n*0.6)
train.id = sample(1:n, ntrain)
train.id
```

```
##  [1]  7 32 39 12 11 13  1 34 40 37 28  4  2 15 23  3  6 24 25 38 42 35 21 16  8
```

```
train.MSE = rep(0, p)
test.MSE = rep(0, p)

train.Y = birthweight[train.id, 2]
train.Y
```

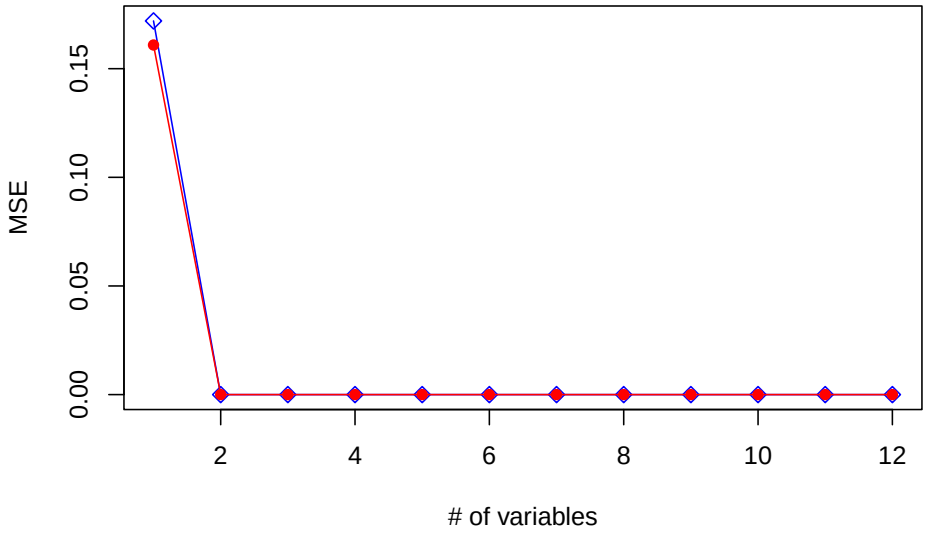
```
##  [1] 3.77 3.32 3.18 3.27 3.35 3.23 4.55 3.87 3.19 3.53 4.57 4.07 4.32 3.15 2.78
## [16] 4.10 3.93 2.51 2.37 3.41 2.75 3.86 3.64 3.11 3.65
```

```
test.Y = birthweight[-train.id, 2]
test.Y
```

```
##  [1] 3.94 3.63 3.42 3.20 3.03 2.92 2.90 2.65 3.14 2.05 1.92 3.59 3.32 3.00 2.74
## [16] 3.55 2.66
```

```
for(i in 1:p){
  myfit = lm( birthweight[train.id, 2]~., birthweight[ train.id, c(1:i, (p+1)) ] )

  train.Y.pred = myfit$fitted
  train.MSE[i] = mean((train.Y - train.Y.pred)^2)


  test.Y.pred = predict(myfit, newdata = birthweight[-train.id, ])
  test.MSE[i] = mean((test.Y - test.Y.pred)^2)
```

```
}

plot(c(1, p), range(train.MSE, test.MSE), type="n", xlab="# of variables", ylab="MSE")
points(train.MSE, col = "blue", pch = 5)
lines(train.MSE, col = "blue", pch = 5)
points(test.MSE, col = "red", pch = 16)
lines(test.MSE, col = "red", pch = 16)
```



In this example both errors are very small which makes it hard to illustrate the differences in the two errors, but feel free to use this code in other examples to observe more patterns between training/testing errors.

In general, we expect that in each iteration, the blue line (representing the training error) consistently exhibits a monotonically decreasing trend, signifying the reduction of training error as more predictors are incorporated. On the other hand, the red line's (testing error) trajectory may not uniformly decrease.

2.11.0.11 Test for Outliers

In this example, we check the data set for unusual observations. Specifically, we will check for:

To identify outliers, we use the Bonferroni test. For that purpose, we need to compute the studentized residuals t_i (an externally standardized version of residuals). Under the null hypothesis H_0, $t_i \sim t_{n-p}$, where n is the sample size and p is the number of predictors without the intercept. We perform this test for all n observations testing case at level $\frac{\alpha}{n}$:

```
## We work with the full model again.
n = dim(birthweight)[1];
```

```
p = length(variable.names(birthweight.mlr.full));
```

We first compute the studentized residuals using the `rstudent` R function and the Bonferroni critical value using Student's distribution:

```
## Computing Studentized Residuals #
birthweight.resid = rstudent(birthweight.mlr.full);

## Critical value WITH Bonferroni correction #
bonferroni_cv = qt(.05/(2*n), n-p-1)
bonferroni_cv
```

```
## [1] -3.60755
```

Now, we need to find which (if any) studentized residuals exceed the Bonferroni critical value:

```
## Sorting the residuals in descending order to find outliers (if any)
birthweight.resid.sorted = sort(abs(birthweight.resid), decreasing=TRUE)[1:10]
print(birthweight.resid.sorted)
```

```
##        2       29       34        1       28       20       42       24
## 2.068621 1.890299 1.830714 1.682375 1.645611 1.613154 1.435773 1.411769
##       19       39
## 1.347956 1.324123
```

```
birthweight.outliers = birthweight.resid.sorted[abs(birthweight.resid.sorted) > abs(bo
print(birthweight.outliers)
```

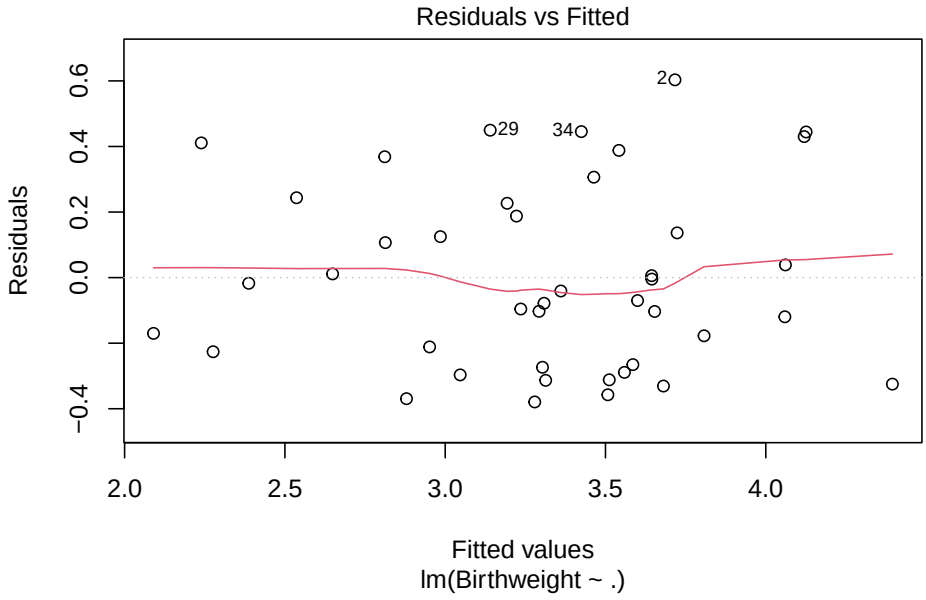
```
## named numeric(0)
```

Above, we computed a t-value of $|-3.61|$ at $\alpha = 0.05$. If an observation's studentized residual is higher (in absolute value) than the critical value of the T distribution with Bonferroni correction, then this observation will be considered an outlier. According to this criterion, we can see that **we don't have any outliers in the data set, since none of the studentized residuals is higher than** $|-3.61|$**.**

2.11.0.12 Checking Model Assumptions

We start by using residual plots and specifically the residuals against fitted values:

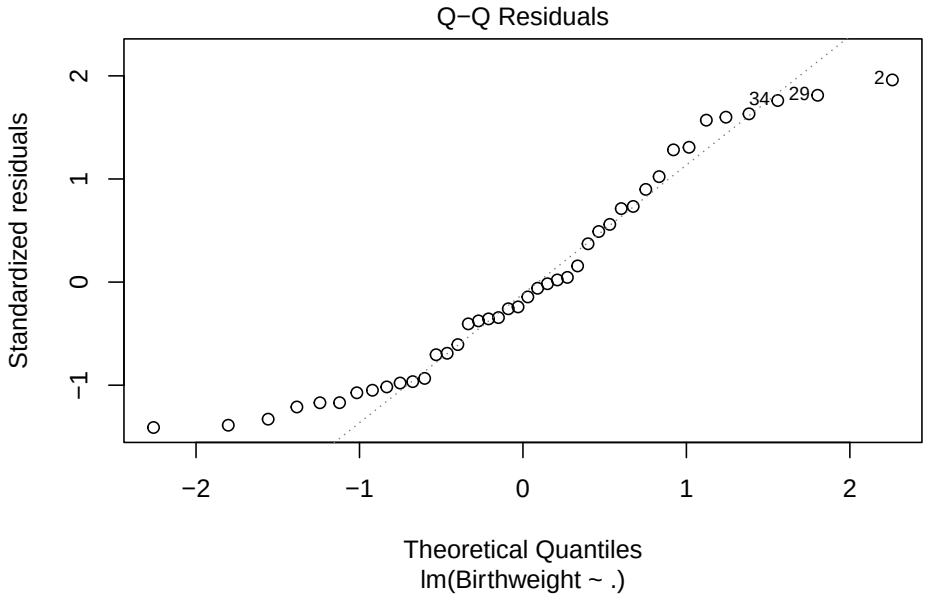
```
plot(birthweight.mlr.full, which=1)
```



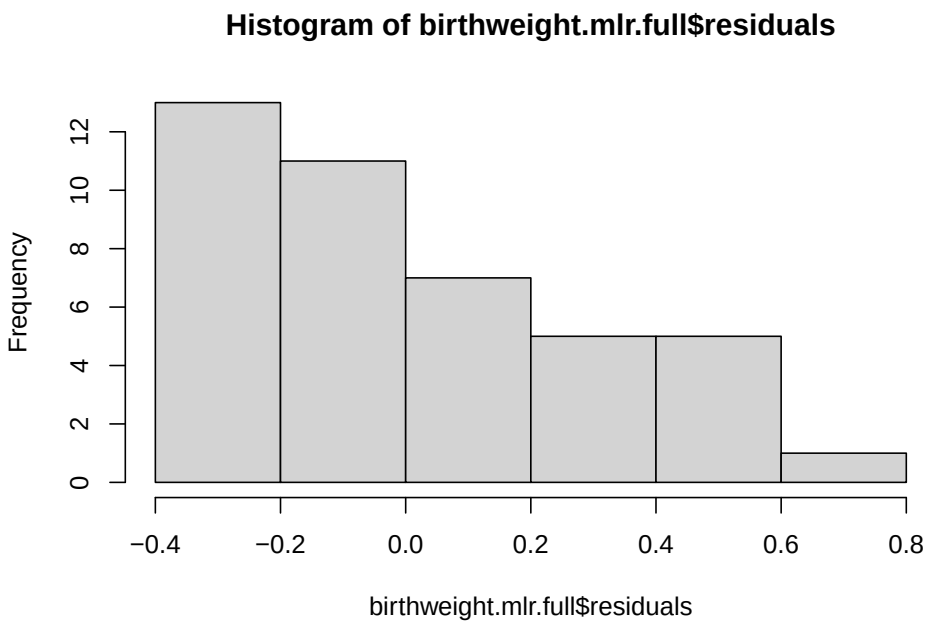
The points on the plot are randomly scattered around the zero line, so we conclude that the constant variance assumption is satisfied.

We continue by checking the normality assumption:

```
plot(birthweight.mlr.full, which=2)
```



```
hist(birthweight.mlr.full$residuals)
```



We seem to have some departures from the normality assumption in this case.

2.11.0.13 Collinearity: Car Seat Position Data (a Faraway data set)

Car drivers like to adjust the seat position for their own comfort. Car designers would find it helpful to know how different drivers will position the seat depending on their *size* and *age*. Researchers at the HuMoSim laboratory at the University of Michigan collected the following data on **38** drivers:

`Age`: Drivers age in years

`Weight`: Drivers weight in lbs

`HtShoes`: height with shoes in cm

`Ht`: height without shoes in cm

`Seated`: seated height in cm

`Arm`: lower arm length in cm

`Thigh`: thigh length in cm

`Leg`: lower leg length in cm

`hipcenter`: horizontal distance of the midpoint of the hips from a fixed location in the car in mm

To read the data we need to load the `faraway` library and then open the `seatpos` data set.

```
library(faraway)
data(seatpos)
attach(seatpos)
```

Let's start by fitting the **full** model with `hipcenter` as a response and everything else as a predictor:

```
# Fit the FULL model
position.full=lm(hipcenter ~ ., seatpos)
summary(position.full)
```

```
##
## Call:
## lm(formula = hipcenter ~ ., data = seatpos)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -73.827 -22.833  -3.678  25.017  62.337
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 436.43213  166.57162   2.620   0.0138 *
## Age           0.77572    0.57033   1.360   0.1843
## Weight        0.02631    0.33097   0.080   0.9372
## HtShoes      -2.69241    9.75304  -0.276   0.7845
## Ht            0.60134   10.12987   0.059   0.9531
## Seated        0.53375    3.76189   0.142   0.8882
## Arm          -1.32807    3.90020  -0.341   0.7359
## Thigh        -1.14312    2.66002  -0.430   0.6706
## Leg          -6.43905    4.71386  -1.366   0.1824
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 37.72 on 29 degrees of freedom
## Multiple R-squared:  0.6866, Adjusted R-squared:  0.6001
## F-statistic:  7.94 on 8 and 29 DF,  p-value: 1.306e-05
```

Note that the p-value of the overall F test is very small (1.306e-05), which means that *at least one of the predictors is statistically significant*, but at the same time none of the predictors is statistically significant according to the individual t tests. This is strange and definitely a *red flag* that we may have collinearity issues.

To investigate the presence of high correlated predictors as well as the linear correlation of the response with each of the predictors, we start by checking the

correlation matrix:

```
## We use the round function with 2 digits, to round the numbers in the output
## This only affects the printed numbers, not the ones R has stored.
round(cor(seatpos), dig=2)
```

```
##             Age Weight HtShoes    Ht Seated   Arm Thigh   Leg hipcenter
## Age        1.00   0.08   -0.08 -0.09  -0.17  0.36  0.09 -0.04      0.21
## Weight     0.08   1.00    0.83  0.83   0.78  0.70  0.57  0.78     -0.64
## HtShoes   -0.08   0.83    1.00  1.00   0.93  0.75  0.72  0.91     -0.80
## Ht        -0.09   0.83    1.00  1.00   0.93  0.75  0.73  0.91     -0.80
## Seated    -0.17   0.78    0.93  0.93   1.00  0.63  0.61  0.81     -0.73
## Arm        0.36   0.70    0.75  0.75   0.63  1.00  0.67  0.75     -0.59
## Thigh      0.09   0.57    0.72  0.73   0.61  0.67  1.00  0.65     -0.59
## Leg       -0.04   0.78    0.91  0.91   0.81  0.75  0.65  1.00     -0.79
## hipcenter  0.21  -0.64   -0.80 -0.80  -0.73 -0.59 -0.59 -0.79      1.00
```

We observe that the response is *highly correlated with most of the variables* which justifies the low p-value in the overall F test. However, we have **highly correlated predictors** (e.g. Corr(`Leg`, `HtShoes`)=0.91) which means that we will have *collinearity*.

Let's look at the **condition number** of the $\mathbf{X}^T\mathbf{X}$ matrix. Recall that the condition number is *not scale invariant*. This implies that calculating it before standardizing the matrix might lead to incorrect conclusions. So, let's standardize it first:

```
## Extract the design matrix and remove the column of 1s that corresponds to the inter
x = model.matrix(position.full)[,-1]

## Standardize the matrix
x = x - matrix(apply(x,2, mean), 38,8, byrow=TRUE)
x = x / matrix(apply(x, 2, sd), 38,8, byrow=TRUE)
apply(x,2,mean)
```

```
##           Age        Weight       HtShoes            Ht        Seated
## -5.843279e-18  2.534522e-16  9.524545e-16  1.577685e-16 -1.079546e-15
##           Arm         Thigh           Leg
## -1.285521e-16  9.860533e-17 -1.029878e-16
```

```
apply(x,2,var)
```

```
##     Age  Weight HtShoes      Ht  Seated     Arm   Thigh     Leg
##       1       1       1       1       1       1       1       1
```

Compute the condition number using the standardized matrix:

```
## Compute the eigenvalues of the matrix
eigenvalues.x = eigen(t(x) %*% x)
eigenvalues.x$val
```

```
## [1] 209.90786979  45.76108236  17.15850736   8.91545889   7.18612386
## [6]   5.14944541   1.86274750   0.05876483
```

```
## Compute the condition number:
sqrt(eigenvalues.x$val[1]/eigenvalues.x$val[8])
```

```
## [1] 59.7662
```

The condition number is **59.77**, larger than 30, so we conclude that *collinearity is present*.

Let's also check the **Variance Inflation Factor (VIF)**:

```
## Variance Inflation Factor (VIF)
round(vif(x), dig=2)
```

```
##     Age  Weight HtShoes      Ht  Seated     Arm   Thigh     Leg
##    2.00    3.65  307.43  333.14    8.95    4.50    2.76    6.69
```

```
sqrt(307.43)
```

```
## [1] 17.53368
```

Note that the *standard error* for the coefficient associated with `HtShoes` is **17.5 times larger** than it would have been without collinearity.

The next step is to **investigate which variable to remove**. To do so, we study the *pairwise correlations* and perform *partial F-tests* as follows:

```
cor(Seated+Thigh, Ht)
```

```
## [1] 0.9389819
```

```
cor(Seated+Leg, Ht)
```

```
## [1] 0.965607
```

```
cor(Seated+Arm, Ht)
```

```
## [1] 0.9465523
```

```
position.red1 = lm(hipcenter ~ Age + Weight + Ht + Seated, data=seatpos)
summary(position.red1)
```

```
##
## Call:
## lm(formula = hipcenter ~ Age + Weight + Ht + Seated, data = seatpos)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -90.869 -21.163  -3.144  26.773  59.423
##
## Coefficients:
```

```
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 478.65890  159.73362   2.997  0.00515 **
## Age           0.58396    0.42573   1.372  0.17943
## Weight       -0.01535    0.31640  -0.049  0.96159
## Ht           -4.99025    1.64389  -3.036  0.00466 **
## Seated        2.04632    3.41283   0.600  0.55287
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 36.83 on 33 degrees of freedom
## Multiple R-squared:  0.6599, Adjusted R-squared:  0.6186
## F-statistic: 16.01 on 4 and 33 DF,  p-value: 2.224e-07
```

```
position.red2 = lm(hipcenter ~ Ht, data=seatpos)
summary(position.red2)
```

```
##
## Call:
## lm(formula = hipcenter ~ Ht, data = seatpos)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -99.956 -27.850   5.656  20.883  72.066
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 556.2553    90.6704   6.135 4.59e-07 ***
## Ht           -4.2650     0.5351  -7.970 1.83e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 36.37 on 36 degrees of freedom
## Multiple R-squared:  0.6383, Adjusted R-squared:  0.6282
## F-statistic: 63.53 on 1 and 36 DF,  p-value: 1.831e-09
```

```
anova(position.red2, position.red1)
```

```
## Analysis of Variance Table
##
## Model 1: hipcenter ~ Ht
## Model 2: hipcenter ~ Age + Weight + Ht + Seated
##   Res.Df   RSS Df Sum of Sq      F Pr(>F)
## 1     36 47616
## 2     33 44774  3    2841.6 0.6981 0.5599
```

Based on the F test provided in the ANOVA table, we conclude that *the reduced model with* `Ht` *as the only variable is better than the model that includes* `Age`,

Weight, *Ht* *and* *Seated*.

Chapter 3

Variable Selection & Regularization

Let's consider the multiple linear regression model with p predictors **plus** the intercept, i.e.

$$Y \sim 1 + X_1 + X_2 + ... + X_p$$

In many applications, the number of explanatory variables, i.e., p is large while in some cases we could even have $p \gg n$. However, this does not necessarily mean that all the variables are relevant to the response Y. In fact, *only a small portion* of the p variables are believed to be relevant to Y.

Our **goal** in this chapter is to develop methods that will allow us to efficiently identify the set of predictors that are useful in estimating or predicting the response. Since the least squares estimator $\hat{}$ is *unbiased*, this implies that irrelevant estimators $\widehat{\beta}_j$ will eventually go to zero. So, if our task is to do well on prediction, *then we need to reflect on whether it is it important to remove unnecessary variables from the model.*

To better understand the implications of unnecessary parameters in a MLR model, we further discuss and quantify the **Training** and **Testing Errors** .

3.1 Training vs. Testing Errors

Consider that we *split our data in two parts*:

- **Training data** $\{\mathbf{x}_i, y_i\}_{i=1}^n$ are used to fit our model
- **Testing data** $\{\mathbf{x}_i, y_i^*\}_{i=1}^n$ are an **independent** data set collected at locations $\mathbf{x}_i$

Remark: In practice, we are given a full data set to analyze, which we then need to split it in two parts (typically in a random fashion) - a *training* part and a *testing* part with a higher percentage allocated to the training data (usually ~80%).

We assume that both *testing* and *training* data come from the same population or are collected at the same locations $\mathbf{x}_i$. Then, *statistically* we can write both models as follows:

$$\mathbf{y}_{n\times 1},\ \mathbf{y}^*_{n\times 1} \sim^{iid} N_n(\ ,\sigma^2\mathbf{I}_n) \text{ and } \quad = \mathbf{X}$$

or equivalently,

$$\mathbf{y} = \mathbf{X}\ +\ ,\quad \sim^{iid} \mathcal{N}_n(\mathbf{0},\sigma^2\mathbf{I}_n) \quad \text{and} \quad \mathbf{y}^* = \mathbf{X}\ +\ ^*,\ ^* \sim^{iid} \mathcal{N}_n(\mathbf{0},\sigma^2\mathbf{I}_n)$$
$$\text{with} \quad ,\ ^* \text{ independent.}$$

Having these models in mind, we compute the **MSE for train and testing data**:

Note that for simplicity in the calculations below, we have suppressed the intercept (so there is no β_0). If we have an intercept, then instead of p, we will have $p+1$.

$$\begin{aligned}
\mathbb{E}(\text{Train Error}) &= \mathbb{E}\Big\|\mathbf{y}-\hat{\mathbf{y}}\Big\|^2 = \mathbb{E}\Big\|(\mathbf{I}-\mathbf{H})\mathbf{y}\Big\|^2 \\
&= tr\Big((\mathbf{I}-\mathbf{H})Cov(\mathbf{y})(\mathbf{I}-\mathbf{H})^T\Big) \\
&= \sigma^2 tr\Big((\mathbf{I}-\mathbf{H})\Big) = (n-p)\sigma^2 \\
&= n\sigma^2 - p\sigma^2
\end{aligned}$$

$$\begin{aligned}
\mathbb{E}(\text{Test Error}) &= \mathbb{E}\Big\|\mathbf{y}^*-\mathbf{X}\hat{}\Big\|^2 \\
&= \mathbb{E}\Big\|(\mathbf{y}^*-\mathbf{X}\ )+(\mathbf{X}\ -\mathbf{X}\hat{})\Big\|^2 \\
&= \mathbb{E}\Big\|\mathbf{y}^*-\ \Big\|^2 + \mathbb{E}\Big\|\mathbf{X}\ -\mathbf{X}\hat{}\Big\|^2 \\
&= \mathbb{E}||\ ^*||^2 + tr\Big(\mathbf{X}Cov(\hat{})\mathbf{X}^T\Big) \\
&= n\sigma^2 + \sigma^2 tr\mathbf{H} \\
&= n\sigma^2 + p\sigma^2
\end{aligned}$$

From the previous equations we can conclude that:

- the *training error decreases with* p .
- the *testing error increases with* p .

This implies that if our goal is *pure prediction*, adding more variables to matrix $\mathbf{X}$ is not the best option. But, *does this imply that the intercept-only model with* $p = 0$*, i.e. the one with the smallest expected test error is the best*? **No!**

The previous analysis is based on the following **Assumptions**:

1. The mean of $\mathbf{y}$, i.e. $E(\mathbf{Y}|\mathbf{X})$, is in $\mathcal{C}(\mathbf{X})$, i.e., there exists some coefficient vector such that $E(\mathbf{Y}|\mathbf{X}) = \mathbf{X}$.
2. The design matrix $\mathbf{X}$ above contains all available predictors. But, when we run a linear regression model using only a **subset** of the columns of $\mathbf{X}$, there will be *an additional* ***Bias*** *term.*

Even more generally, we may have a *model*:

$$Y = f(X) + \varepsilon$$

in which f is not even linear. In such cases, we approximate f with a linear function f^* which means that we introduce bias. (Intuitively, this means that usually when we do not know the underlying true model, a linear approximation is often a good first step.) In the same spirit, even when the underlying function is truly linear, we may still introduce bias when we use a subset of the design matrix $\mathbf{X}$, i.e. when the model misses some relevant variables.

In such cases, the training and testing errors compute as:

$$\mathbb{E}(\text{Test Error})^2 = n\sigma^2 + p\sigma^2 + Bias$$
$$\mathbb{E}(\text{Training Error})^2 = n\sigma^2 - p\sigma^2 + Bias$$

where $p\sigma^2$ is the unavoidable error – the ε – and Bias is the model error.

- Bigger model (i.e., p large) $\rightarrow$ small Bias, but large Variance ($p\sigma^2$)
- Smaller model (i.e., p small) $\rightarrow$ large Bias, but small Variance ($p\sigma^2$).

To reduce the *test error* (i.e., prediction error), the key is to find the best **trade-off** between Bias and Variance.

3.2 Subset Selection

The idea behind *subset selection* is to score each model according to an information criterion and then use a search algorithm to find the optimal model. In this way, we take into account

Training Error + Complexity Penalty

In the context of linear regression models, the *complexity of a model increases with the number of predictor variables* (i.e., p).

- **Training Error**: an increasing function of RSS.
- **Complexity Penalty**: an increasing function of p.

  Why don't we just use R^2 or RSS? The main reason is that R^2 always increases when we introduce variables in the model, while RSS always reduces. Therefore, R^2 and RSS do not penalize for introducing unnecessary variables in the model.

3.2.1 Information Criteria-based procedures

Akaike Information Criterion & Bayesian Information Criterion

AIC and BIC are defined as

$$AIC = -2 \cdot loglik + 2\ p$$
$$BIC = -2 \cdot loglik + \log(n)\ p$$

where p is the number of predictors included in model under consideration. $loglik$ denotes the log-likelihood of the model under consideration. For the normal-error linear regression model, the first term computes:

$$-2 \cdot loglik = n \log \frac{RSS}{n}$$

which means that

$$AIC = n \log \frac{RSS}{n} + 2\ p$$
$$BIC = n \log \frac{RSS}{n} + \log(n)\ p$$

The **lower** the AIC/BIC the **better**. Note that when n is large, *adding an additional predictor costs a lot more in BIC than AIC*. So, AIC tends to pick a bigger model than BIC.

Adjusted-R^2 for model with p predictors

$$\begin{aligned} R_a^2 &= 1 - \frac{RSS/(n-p-1)}{TSS/(n-1)} \\ &= 1 - (1-R^2)\Big(\frac{n-1}{n-p-1}\Big) \\ &= 1 - \frac{\hat{\sigma}^2}{\hat{\sigma}_0^2} \end{aligned}$$

where $\hat{\sigma}$ is the estimated error variance for the current fitted model, and $\hat{\sigma}_0^2$ is the estimated error variance for the full fitted model.

The **higher** the R_a^2 the **better**.

Mallow's C_p

$$C_p = \frac{RSS_{\mathbf{p}}}{\hat{\sigma}_0^2} + 2p - n$$

where $\hat{\sigma}_0^2$ is the estimate of the error variance for the full model. Mallow's C_p behaves very similar to AIC and the **lower** the C_p the **better**.

Illustration of Complexity Criteria vs. p

As an illustration, the complexity penalties R^2_{adjusted}, Mallow's C_p, AIC, and BIC are plotted as a function of p (the number of parameters in the model) for the `student-grades` data set (the example in the last section):

3.2.2 Search Algorithms

In the beginning of this section, we discussed that to select the optimal model, we need to penalize the models using the preferred penalty. If we have (in total) p available predictors, then there are 2^p potential models. Ideally, we would like to search and score all of these models. If p is large, then we need to employ systematic algorithms that search and score potential models.

Level-wise search algorithms: These are algorithms return the global optimal solution among *all possible models* and work only for less than *40* variables.

The idea is that they finds m different models (default m in `R` is 8) of up to size p with the smallest RSS among all the models of the same size. Then, they evaluate the score on the p models and report the optimal one. It is important to note that the algorithm does not need to visit every model. For example, if we know that $RSS(X1, X2) < RSS(X3, X4, X5, X6)$, then it is not necessary to search any size 2 or 3 *sub-models* of set $(X3, X4, X5, X6)$ meaning that these models may be leaped over.

Greedy algorithms: These algorithms add and/or remove variables based on the score given by a criterion such as AIC/BIC. They can move :

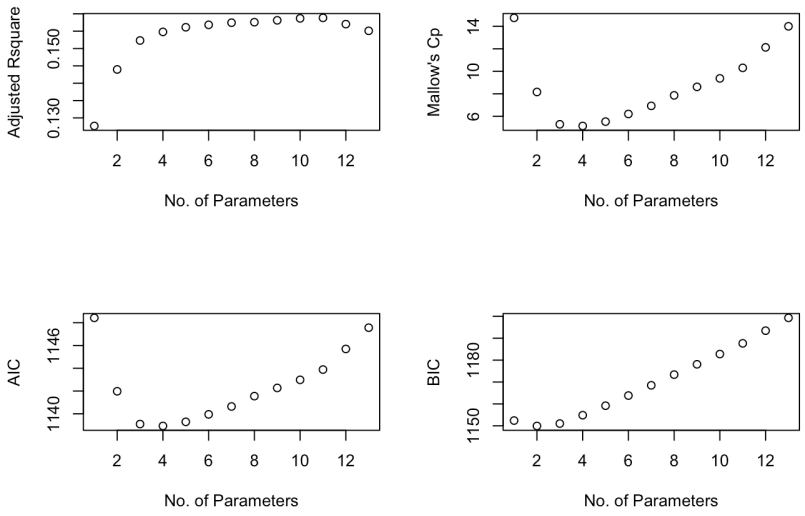


Figure 3.1: Subset selection criteria vs. p.

- *Backwards*: start with the full model and sequentially delete predictors until the score does not improve.
- *Forward*: start with the null model and sequentially add predictors until the score does not improve.
- *Stepwise*: consider both deleting and adding one predictor at each stage.

These algorithms are computationally efficient, but they only return a *locally optimal* solution which usually good enough in practice.

Some considerations

In some cases, specially when $p \gg n$ starting with the full model and using the stepwise procedure is not feasible. Therefore, we need to screen some of the variables and remove them, before employing the search algorithms. A common variable screening approach is the following:

- Pick a *smaller initial model* as a starting point by **ranking** the p predictors by the absolute value of their (marginal) correlation with Y.
- Keep the **top** K predictors (e.g., $K = n/3$).
- Use the stepwise procedure so that you can add removed variables back to the model.

3.3 Shrinkage Methods

The variable selection methods we already discussed work quite well in practice when it comes to data sets where we do not have too many predictors. For example, recall that the *level-wise* algorithms are good when $p < 40$.

In this section, we are going to study two different methods we can use to **shrink** the number of predictors in order to select the optimal model that balances the *trade-off between model bias and prediction error (variance).*

Let us then re-frame what we have been doing so far:

Regression with Penalization

$$\begin{aligned}\hat{\beta} &= \arg\min_{\beta} ||\mathbf{y} - \mathbf{X}\beta||^2 + \underbrace{\lambda \sum_{j=1}^{p} \mathbf{1}_{\beta_j \neq 0}}_{\text{penalty}} \\ &= \arg\min_{\beta} ||\mathbf{y} - \mathbf{X}\beta||^2 + \underbrace{\lambda ||\beta||_0}_{\text{penalty}}\end{aligned}$$

where $||\beta||_0 = \sum_{j=1}^{p} \mathbf{1}_{\beta_j \neq 0}$. The *vector norm* $||\beta||_0$ can be thought of as a function that decides whether a variable is **in** or **out** of the model. Then, the different criteria we discussed before rise by choosing λ properly.

Taking this idea one step further (remembering our linear algebra tools), we can replace the *0-norm* above with a different one, e.g. the L^1 or the L^2 norm. Exactly this observation, lead to two extremely popular *penalized regression* methods:

- the **Ridge Regression**

$$\hat{\beta} = \arg\min_{\beta} ||\mathbf{y} - \mathbf{X}\beta||^2 + \underbrace{\lambda ||\beta||^2}_{\text{penalty}}, \text{ where } ||\beta||^2 = \sum_{j=1}^{p} \beta_j^2$$

- the **Lasso Regression**

$$\hat{\beta} = \arg\min_{\beta} ||\mathbf{y} - \mathbf{X}\beta||^2 + \underbrace{\lambda |\beta|}_{\text{penalty}}, \text{ where } |\beta| = \sum_{j=1}^{p} |\beta_j|$$

In the next sections, we discuss the mathematics and the implementation of these methods. However, before proceeding, we need to make two consider the following:

Implementation Considerations

When the penalization norm changes, we observe that the penalty term that arises (in both cases) *is not location or scale invariant.* This implies that if

we re-scale a variable then the penalized β estimators will be sensitive to these values. Therefore, it is suggested that we **center** and **scale** each column of the design matrix $\mathbf{X}$. Specifically, if $\mathbf{x}_j$, $j = 1, \dots, p$ is a column of $\mathbf{X}$, then

$$\tilde{\mathbf{x}}_j = \frac{\mathbf{x}_j - \bar{\mathbf{x}}_j}{sd_{\mathbf{x}_j}}$$

In addition to that we **center y** to *suppress the intercept*, i.e.

$$\tilde{\mathbf{y}}_i = \mathbf{y}_i - \bar{\mathbf{y}}$$

After the model selection/fitting, we can always transform the variables back to their original values so that we continue with interpretation and/or prediction. We can also *estimate back the intercept* as follows:

$$\hat{\beta}_0 = \bar{y} - \sum_{j=1}^{n} \hat{\beta}_j \bar{\mathbf{x}}_j$$

Note that in `R` tha `glmnet` package handles the centering and scaling (and transformation back to original) automatically.

3.3.1 Ridge Regression

Ridge regression assumes that after normalization, some of the regression coefficients should not be very large. Ridge regression is very useful when you have collinearity and the LS regression coefficients are unstable. In fact, it was initially introduced by A. Tikhonov to remedy multi-collinearity problems by adding a non-negative constant to the diagonal of the design matrix.

Ridge Regression

The idea of the method is to add a **penalty** term to the LS minimization problem :

$$\text{minimize}_\beta \, (y - X\beta)^T (y - X\beta) + \lambda \sum_j \beta_j^2$$

for some $\lambda \geq 0$. The penalty term is $\sum_j \beta_j^2$.

As discussed before, for the method to be more effective, we prefer to *standardize* the predictors first (centered by their means and scaled by their standard deviations) and center the response y.

One of the main advantages of the ridge regression is that it provides us with closed-form solutions for the β coefficients. Indeed, solving the minimization problem we obtain:

$$\hat{\beta}_{\text{Ridge}} = (X^T X + \underbrace{\lambda I}_{\text{ridge}})^{-1} X^T y$$

It is easy to see that when $\lambda = 0$ the ridge regression estimation problem reduces to the standard least squares problem, while when $\lambda \to \infty$, the ridge coefficients $\hat{\beta} \to \mathbf{0}$. As a result, during implementation, one of the main considerations is how to choose λ. Choosing a very small (relatively speaking) value for λ leads back to the usual LS estimators, while choosing a very large value for λ makes most estimators zero.

To balance this trade-off in practice, a common approach is to use automated methods such as *Generalized Cross-Validation* (GCV). The main *disadvantage* of the ridge regression estimators is that they are **biased**, that is $\mathbb{E}(\hat{\beta}) = \beta +$ bias.

To better understand the structure of the ridge regression LS Coefficients, assume that $\mathbf{X}^T\mathbf{X} = \mathbf{I}_p$, that is the columns of the design matrix are orthogonal. Then, the general formula above reduces to

$$\hat{\beta}_{ridge} = \left(\mathbf{X}^T\mathbf{X} + \lambda I\right)^{-1}\mathbf{X}^T\mathbf{y} = \frac{1}{1+\lambda}\mathbf{X}^T\mathbf{y}$$

Similarly, the fitted values

$$\hat{\mathbf{y}}_{ridge} = \mathbf{X}\hat{\beta}_{ridge} = \frac{1}{1+\lambda}\,\hat{\mathbf{y}}_{LS}$$

If the columns of the design matrix are not orthogonal, then we can run the regression against an orthonormal version of $\mathbf{X}$, known as *principal components analysis*, or *singular value decomposition*.

Singular Value Decomposition (SVD)

The Singular Value Decomposition (SVD) is one of the most important concepts in applied mathematics. It is used for a number of application including dimension reduction and data analysis. Principal Components Analysis (PCA) is also a special case of the SVD.

SVD of $\mathbf{X}_{n\times p}$

Consider the design matrix $\mathbf{X}_{n\times p}$. Then, $\mathbf{X}$ can be written as

$$\mathbf{X} = \mathbf{U}_{n\times p}\,\mathbf{D}_{p\times p}\,\mathbf{V}^T_{p\times p}$$

where

- $\mathbf{U}_{n\times p}$ orthogonal matrix with columns that are spanning $\mathcal{C}(\mathbf{X})$.
- $\mathbf{V}_{p\times p}$ orthogonal matrix with columns that are spanning $\mathbb{R}^p$.
- $\mathbf{D}_{p\times p}$ diagonal values with $d_1 \geq ... \geq d_p \geq 0$ the singular values of $\mathbf{X}$. If one or more $d_j = 0$, then $\mathbf{X}$ is singular, i.e. not full-rank.

This factorization of the matrix $\mathbf{X}$ is called the **singular value decomposition of X**, and the columns of $\mathbf{U}$ and $\mathbf{V}$ are called the left- and right-hand **singular vectors** of $\mathbf{X}$.

For simplicity in the discussion, let $n > p$ and $rank(\mathbf{X}) = p$, which also implies that $d_p > 0$.

Some Useful Properties of the SVD

1. The left-hand singular vectors are a set of orthonormal eigenvectors for $\mathbf{X}^T\mathbf{X}$, i.e. $\mathbf{U}^T\mathbf{U} = \mathbf{I}$.

2. The right-hand singular vectors are a set of orthonormal eigenvectors for $\mathbf{X}\mathbf{X}^T$, i.e. $\mathbf{V}^T\mathbf{V} = \mathbf{I}$.

3. The eigenvectors $\mathbf{v}_j$ are also called the *principal components directions* of the columns of $\mathbf{X}$.

4. The first principal component direction v_1 has the property that $z_1 = \mathbf{X}\mathbf{v}_1 = \mathbf{u}_1 d_1$ has the largest sample variance among all normalized linear combinations of the columns of $\mathbf{X}$.

5. The singular values are the square roots of the eigenvalues for $\mathbf{X}^T\mathbf{X}$ and $\mathbf{X}\mathbf{X}^T$, since these matrices have the same eigenvalues.

6. The first singular value is equal to

$$\sigma_1 = \max_{||x||=1} ||\mathbf{X}||_2$$

One of the goals of *principal components analysis* is to find the new coordinates, or *scores*, of the data in the principal components basis. If the original (centered or standardized) data was contained in the matrix $\mathbf{X}$ and the eigenvectors of the covariance/correlation matrix ($\mathbf{X}^T\mathbf{X}$) were columns of a matrix $\mathbf{V}$, then to find the scores ($\mathbf{S}$) of the observations on the eigenvectors we can use the following equation:

$$\mathbf{X} = \mathbf{S}\mathbf{V}^T$$

where each columns of $\mathbf{S}_{n\times p} = \mathbf{UD}$ is the so-called **principal component** and each column of V is the **principal component direction** of $\mathbf{X}$.

SVD of Fitted Values: LS vs. Ridge

To observe how the *ridge* affects the structure of the fitted β, we have the following.

- In *least-squares* regression, the fitted values compute as:

$$\begin{aligned}
\hat{\mathbf{y}}_{LS} &= \mathbf{X}\hat{\beta}_{LS} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \\
&= \mathbf{U}\,\mathbf{D}\,\mathbf{V}^T(\mathbf{V}\,\mathbf{D}^2\,\mathbf{V}^T)^{-1}\mathbf{V}\,\mathbf{D}\,\mathbf{U}^T\mathbf{y} \\
&= \mathbf{U}\,\mathbf{D}\,\mathbf{V}^T(\mathbf{V}^T)^{-1}\,\mathbf{D}^{-2}\,\mathbf{V}^{-1}\mathbf{V}\,\mathbf{D}\,\mathbf{U}^T\mathbf{y} \\
&= \mathbf{U}\,\mathbf{D}\,\mathbf{D}^{-2}\mathbf{D}\,\mathbf{U}^T\mathbf{y} \\
&= \mathbf{U}\mathbf{U}^T\mathbf{y} \\
&= \sum_{j=1}^{p}(\mathbf{u}_j^T\mathbf{y})\mathbf{u}_j
\end{aligned}$$

where we used the facts that $\mathbf{U}^T\mathbf{U} = \mathbf{I}$, and that

$$\mathbf{X}^T\mathbf{X} = \mathbf{V}\,\mathbf{D}^T\,\mathbf{U}^T\mathbf{U}\,\mathbf{D}\,\mathbf{V}^T = \mathbf{V}\,\mathbf{D}^2\,\mathbf{V}^T$$

The last expression is the *eigen-decomposition of* $\mathbf{X}^T\mathbf{X}$.

Similarly, for the ridge regression coefficients, we have:

$$\begin{aligned}
\hat{\mathbf{y}}_{ridge} &= \mathbf{X}\hat{\beta}_{ridge} \\
&= \mathbf{X}(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \\
&= \mathbf{U}\,\mathbf{D}\,\mathbf{V}^T(\mathbf{V}\,\mathbf{D}^2\,\mathbf{V}^T + \lambda\mathbf{I})^{-1}\mathbf{V}\,\mathbf{D}\,\mathbf{U}^T\mathbf{y} \\
&= \mathbf{U}\,\mathbf{D}\,\mathbf{V}^T(\mathbf{V}\,\mathbf{D}^2\,\mathbf{V}^T + \lambda\mathbf{V}\mathbf{V}^T)^{-1}\mathbf{V}\,\mathbf{D}\,\mathbf{U}^T\mathbf{y} \\
&= \mathbf{U}\,\mathbf{D}\,\mathbf{V}^T(\mathbf{V}\,(\mathbf{D}^2 + \lambda\mathbf{I})\,\mathbf{V}^T)^{-1}\mathbf{V}\,\mathbf{D}\,\mathbf{U}^T\mathbf{y} \\
&= \mathbf{U}\,\mathbf{D}\,\mathbf{V}^T(\mathbf{V}^T)^{-1}\,(\mathbf{D}^2 + \lambda\mathbf{I})^{-1}\,\mathbf{V}^{-1}\mathbf{V}\,\mathbf{D}\,\mathbf{U}^T\mathbf{y} \\
&= \mathbf{U}\mathbf{D}\,(\mathbf{D}^2 + \lambda\mathbf{I})^{-1}\,\mathbf{D}\mathbf{U}^T\mathbf{y}
\end{aligned}$$

In this last expression, note that

- $\mathbf{D}\,(\mathbf{D}^2 + \lambda\mathbf{I})^{-1}$ is a diagonal matrix with elements given by $\frac{d_j^2}{d_j^2+\lambda}$.
- the vector $\mathbf{U}^T\mathbf{y}$ is the coordinates of the vector $\mathbf{y}$ in the basis spanned by the p columns of $\mathbf{U}$.

Therefore, $\hat{\mathbf{y}}_{ridge}$ simplifies to

$$\hat{\mathbf{y}}_{ridge} = \sum_{j=1}^{p}\mathbf{u}_j\,\frac{d_j^2}{d_j^2+\lambda}\,\mathbf{u}_j^T\mathbf{y}$$

As we can observe, the inner products $\mathbf{u}_j^T\mathbf{y}$ are scaled by the factors $\frac{d_j^2}{d_j^2+\lambda}$, or in other words the ridge estimate $\hat{\beta}_{ridge}$ **shrinks the LS estimate** $\hat{\beta}_{LS}$ **by**

a factor of $\frac{d_j^2}{d_j^2+\lambda}$ where *the smaller the eigenvalues the more the shrinkage.* Essentially, ridge regression projects $\mathbf{y}$ onto the principal components, and then shrinks the coefficients of the low–variance components more than the high-variance components.

Complexity of Ridge Regression

To quantify the complexity of a model, heuristically we need to understand the number of coefficients that need to be estimated. So, a linear regression model is a model with p covariates and p β coefficients, so it has p degrees of freedom. In ridge regression however, the estimated βs are still p-dimensional, however the method does not use all strength of the p covariates due to shrinkage. If λ is extremely large, there will be no effective covariates left in the model, meaning that they should all be close to zero. On the other hand, if λ is 0, then we go back to linear regression with p covariates and p degrees of freedom. So, in the case of the ridge regression, the truth lies somewhere between 0 and p.

Let's formalize the intuition, by recalling that one method to compute the degrees of freedom of a model is to relate them to the correlation between the observed and fitted values as follows:

$$df = \sum_{i=1}^{n} Cor(y_i,\ \hat{y}_i)$$

- **Linear regression**: We have already shown that the fitted values can be expressed as $\hat{\mathbf{y}} = \mathbf{H}\mathbf{y}$, in which case

$$df = \sum_{i=1}^{n} Cor(y_i,\ \hat{y}_i) = \sum_{i=1}^{n} H_{ii} = tr(\mathbf{H}) = p$$

  where (recall) $\mathbf{H}$ is the hat (projection) matrix equal to $\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$.

- **Ridge regression**: We have shown that $\hat{y} = \underbrace{\mathbf{X}(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T}_{:=\mathbf{S}_\lambda}\mathbf{y} = \mathbf{S}_\lambda\mathbf{y}$. So, the effective degrees of freedom of the ridge regression are

$$\begin{aligned} df(\lambda) &= \sum_{i=1}^{n} Cor(y_i,\ \hat{y}_i) = \sum_{i=1}^{n} [S_\lambda]_{ii} \\ &= tr(\mathbf{S}_\lambda) \\ &= tr(\mathbf{X}(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T) \\ &= tr\left(\sum_{j=1}^{p} \frac{d_j^2}{d_j^2+\lambda}\mathbf{u}_i\mathbf{u}_i^T\right) = \sum_{j=1}^{p} \frac{d_j^2}{d_j^2+\lambda} \end{aligned}$$

Based on the expression above, we can see that the degrees of freedom of the ridge regression are a *decreasing* function of λ, and reduce to p when $\lambda = 0$. One important consequence of this expression is that we can use it to determine the values of λ to use when applying cross validation. This can be done by thinking $df(\lambda)$ as a function of λ and then setting $df(\lambda) = k$, with $k = 1, \dots, p$ representing all possible dfs. Then, we can solve with respect to λ using a numerical approach.

3.3.2 Lasso Regression

Lasso Regression is similar to the Ridge regression in the sense that it minimizes the least squares criterion *subject to a penalty term*. However, the penalty term is different in the case of lasso.

Lasso Regression

$\hat{\beta}_{\text{LASSO}}$ minimizes:

$$\text{minimize } (y - X\beta)^T(y - X\beta) + \lambda \sum_j |\beta_j|$$

for some $\lambda \geq 0$. The penalty term is $\sum_j |\beta_j|$ (L_1 constraint).

In two-dimensions the constraint defines a square, while in higher dimensions it defines a polytope. Lasso is useful when the response can be explained by *few* predictors with zero effect on the remaining predictors (Lasso is similar to a variable selection method). When $\beta_j = 0$ the corresponding predictor is eliminated which is not the case for ridge regression. Therefore, we use lasso when the effect of predictors is **sparse**. This means that only few predictors will have an effect on the response (e.g. gene expression data) or when number of predictors is large ($p > n$).

The lasso solution is defined as

$$\hat{\beta}_{lasso} = \arg\min_{\beta \in \mathbb{R}^p} \Big((y - X\beta)^\top (y - X\beta) + \lambda \sum_j |\beta_j| \Big)$$

and does not have a closed form expression.

If we assume that $\mathbf{X}^T\mathbf{X} = \mathbf{I}_p\}$, then

$$\begin{aligned} ||\mathbf{y} - \mathbf{X}\beta||^2 &= ||\mathbf{y} - \mathbf{X}\hat{\beta}_{LS} + \mathbf{X}\hat{\beta}_{LS} - \mathbf{X}\beta||^2 \\ &= ||\mathbf{y} - \mathbf{X}\hat{\beta}_{LS}||^2 + ||\mathbf{X}\hat{\beta}_{LS} - \mathbf{X}\beta||^2 \end{aligned}$$

where

$$2(\mathbf{y} - \mathbf{X}\hat{\beta}_{LS})^T(\mathbf{X}\hat{\beta}_{LS} - \mathbf{X}\beta) = 2\, r^T\, (\mathbf{X}\hat{\beta}_{LS} - \mathbf{X}\beta) = 0$$

since the swecond term is a linear combination of columns of $\mathbf{X}$ no matter what value β takes, and thus is in $\mathcal{C}(\mathbf{X})$, therefore orthogonal to the residual vector r.

Obtaining the Lasso Solution

Although we do not have a closed form solution in the case of lasso, the minimization problem we have to solve is not very challenging.

The lasso solution can be expressed as:

$$\begin{aligned}
\hat{\beta}_{lasso} &= \arg\min_{\beta\in\mathbb{R}^p}\Big(||\mathbf{y}-\mathbf{X}\beta||^2+\lambda|\beta|\Big)\\
&= \arg\min_{\beta\in\mathbb{R}^p}\Big(||\mathbf{X}\hat{\beta}_{LS}-\mathbf{X}\beta||^2+\lambda|\beta|\Big)\\
&= \arg\min_{\beta\in\mathbb{R}^p}\Big((\hat{\beta}_{LS}-\beta)^T\mathbf{X}^T\mathbf{X}(\hat{\beta}_{LS}-\beta)+\lambda|\beta|\Big)\\
&= \arg\min_{\beta\in\mathbb{R}^p}\Big((\hat{\beta}_{LS}-\beta)^T(\hat{\beta}_{LS}-\beta)+\lambda|\beta|\Big)\\
&= \arg\min_{\beta_1,\dots,\beta_p}\sum_{i=1}^{p}\Big((\beta_j-\hat{\beta}_{(LS)j})^2+\lambda|\beta_j|\Big)
\end{aligned}$$

We can find the *optimal* β_j for each of $j=1,\dots,p$ **separately** by solving the following generic problem for each dimension:

$$\arg\min_x (x-a)^2+\lambda|x|,\ \lambda>0$$

Therefore, to solve the one-dim lasso above, define

$$f(x)=(x-a)^2+\lambda|x|,\ a\in\mathbb{R},\ \lambda>0$$

The value x^* that minimizes $f(x)$ must satisfy:

$$\begin{aligned}
\frac{\partial}{\partial x}(x^*-a)^2+\lambda\,\frac{\partial}{\partial x}|x^*| &= 0\\
2(x^*-a)+\lambda z^* &= 0
\end{aligned}$$

where z^* is the *sub-gradient* of the absolute value function evaluated at x^*, which equals to $sign(x^*)$ if $x^*\neq 0$ and any number in [-1,1], if $x^*=0$.

Therefore, the minimizer of $f(x)$ is given by

$$x^*=S_{\lambda/2}(a)=sign(a)(|a|-\lambda/2)_+=\begin{cases} a-\lambda/2, & \text{if } a>\lambda/2\\ 0, & \text{if } |a|\le\lambda/2\\ a+\lambda/2, & \text{if } a<-\lambda/2\end{cases}$$

where $S_{\lambda/2}(\cdot)$ is often referred to as the **soft-thresholding operator**.

When the design matrix $\mathbf{X}$ is orthogonal, the lasso solution simplifies to

$$\hat{\beta}_j^{lasso} = \begin{cases} sign(\hat{\beta}_{(LS)j} - \lambda/2), \text{ if } |\hat{\beta}_{(LS)j}| > \lambda/2 \\ 0, \text{ if } |\hat{\beta}_{(LS)j}| \leq \lambda/2 \end{cases}$$

A large λ will cause some of the coefficients to be exactly zero. So, lasso does both variable (subset) selection and (soft) shrinkage.

Remarks

- It is suggested to use lasso when the effect of predictors is *sparse*, since lasso will "make" some of the β coefficients zero keeping the coefficients that will have an effect on $\mathbf{y}$. This is the reason why this method also works when the number of predictors is larger than the sample size ($p > n$). These are scenarios often encountered in genomic or proteomic data where the design matrices tend to have a lot of zeros and too many predictors.
- In lasso as in ridge regression, we can select λ using Cross-Validation (CV). When λ increases, the number of predictors decreases.

Comparing Ridge Regression and Lasso

Lasso selects a sub-set of predictors (some coefficients will equal to zero), while ridge regression performs better when the response is a function of many predictors with coefficients around the same size. Lasso will perform better when a relatively small number of predictors have large coefficients and the rest are very small or equal to zero.Since the number of predictors is never known *a priori*, cross-validation can be used to decide which approach is better for a particular data set.

Ridge regression does a proportional shrinkage. Lasso translates each coefficient by a constant factor , truncating at zero.

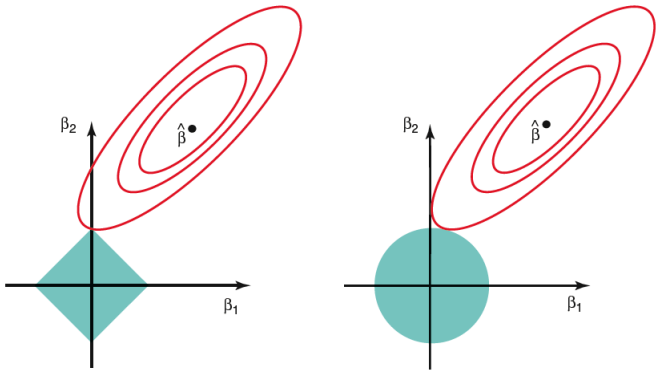


Figure 3.2: Contour of the optimization for Lasso (left) and Ridge (right).

In the picture below, we can see the difference in the contours of optimization for lasso (left) and Ridge (right) when there are only two parameters. The residual sum of squares has elliptical contours, centered at the full least squares estimate, $\widehat{\beta}_{LS}$. The constraint region for ridge regression is the disk $\beta_1^2 + \beta_2^2 \leq t$, while that for lasso is the diamond $|\beta_1| + |\beta_2| \leq t$. Both methods find the first point where the elliptical contours hit the constraint region. Unlike the disk, the diamond has corners which means that if the solution occurs at a corner, then it has one parameter β_j equal to zero. When $p > 2$, the diamond becomes a rhomboid, and has many corners, flat edges and faces; there are many more opportunities for the estimated parameters to be zero.

Lasso with $p > n$

When $\mathbf{X}$ *is of full rank:*

- the lasso solution is the minimizer of a convex function over a convex set
- the lasso solution is unique since the first term is a strictly convex function.

When $\mathbf{X}$ *is not of full rank, or when* $p > n$*:*

- The lasso criterion is no longer convex which means that it may not have a unique minimizer.first term is no longer strictly convex. In the $p > n$ case, the lasso selects at most n variables before it saturates, because of the nature of the convex optimization problem which is a limiting feature for a variable selection method.

3.4 The `Student Performance` Example

The `student-mat.csv` data set contains data from a study on student achievement in secondary education of two Portuguese schools. The data are based on the following publication:

> "Using data mining to predict secondary school student performance"
> By P. Cortez, A. M. G. Silva. 2008. Published in Proceedings of 5th Annual Future Business Technology Conference.

and can be found in the UCI data repository: here

The variables in the data set are the following:

- `school` Student's school (*binary*: `GP` - Gabriel Pereira or `MS` - Mousinho da Silveira)
- `sex` Sex student's sex (*binary*: `F` - female or `M` - male)
- `age` Student's age (*numeric*: from 15 to 22)
- `address` Student's home address type (*binary*: `U` - urban or `R` - rural)
- `famsize` Family size (*binary*: `LE3` - less or equal to 3 or `GT3` - greater than 3)

- `Pstatus` Parent's cohabitation status (*binary*: `T` - living together or `A` - apart)
- `Medu` Mother's education level (*numeric*: `0` - none, `1` - primary education (4th grade), `2` - 5th to 9th grade, `3` - secondary education or `4` - higher education)
- `Fedu` Father's education level (*numeric*: `0` - none, `1` - primary education (4th grade), `2` 5th to 9th grade, `3` - secondary education or `4` higher education)
- `Mjob` Mother's occupation (*nominal*: `teacher`, `healthcare` related, `civil services` (e.g. administrative or police), `at_home` or `other`)
- `Fjob` Father's occupation (*nominal*: `teacher`, `healthcare` related, `civil services` (e.g. administrative or police), `at_home` or `other`)
- `reason` Reason to choose this school (*nominal*: close to `home`, school `reputation`, `course` preference or `other`)
- `guardian` Student's guardian (*nominal*: `mother`, `father` or `other`)
- `traveltime` Home to school travel time (*numeric*: `1` - <15 min., `2` - 15 to 30 min., `3` - 30 min. to 1 hour, or `4` - >1 hour)
- `studytime` Weekly study time (*numeric*: `1` - <2 hours, `2` - 2 to 5 hours, `3` - 5 to 10 hours, or `4` - >10 hours)
- `failures` Number of past class failures (*numeric*: `n` if 1<=n<3, else `4`)
- `schoolsup` Extra educational support (*binary*: `yes` or `no`)
- `famsup` Family educational support (*binary*: `yes` or `no`)
- `paid` Extra paid classes within the course subject (Math or Portuguese) (*binary*: `yes` or `no`)
- `activities` Extra-curricular activities (*binary*: `yes` or `no`)
- `nursery` Attended nursery school (*binary*: `yes` or `no`)
- `higher` Wants to take higher education (*binary*: `yes` or `no`)

- `internet` Internet access at home (*binary*: `yes` or `no`)

- `romantic` With a romantic relationship (*binary*: `yes` or `no`)

- `famrel` Quality of family relationships (*numeric*: from `1` - very bad to `5` - excellent)
- `freetime` Free time after school (*numeric*: from `1` - very low to `5` - very high)

- `goout` Going out with friends (*numeric*: from `1` - very low to `5` - very high)

- `Dalc` Workday alcohol consumption (*numeric*: from `1` - very low to `5` - very high)
- `Walc` Weekend alcohol consumption (*numeric*: from `1` - very low to `5` - very high)
- `health` Current health status (*numeric*: from `1` - very bad to `5` - very good)

- `absences` Number of school absences (*numeric*: from `0` to `93`)

- G1 First period grade (*numeric*: from 0 to 20)
- G2 Second period grade (*numeric*: from 0 to 20)
- G3 Final grade (*numeric*: from 0 to 20, output target)

```
stu_performance0 = read.csv2("data/week3/student-mat.csv", header=TRUE)
```

In our example, we want to focus on predicting the final grade (column G3) using penalized regression methods. A special variation of penalized regression is required when using categorical variables (e.g. *group lasso*), thus in this example we only focus on numerical predictors.

```
## Remove categorical predictors
stu_performance = stu_performance0[,-c(1,2,4,5,6,9,10, 11, 12, 16, 17, 18, 19, 20, 21,

## Rename the response to "Grades"
names(stu_performance)[14] = 'Grades'

head(stu_performance)
```

```
##   age Medu Fedu traveltime studytime failures famrel freetime goout Dalc Walc
## 1  18    4    4          2         2        0      4        3     4    1    1
## 2  17    1    1          1         2        0      5        3     3    1    1
## 3  15    1    1          1         2        3      4        3     2    2    3
## 4  15    4    2          1         3        0      3        2     2    1    1
## 5  16    3    3          1         2        0      4        3     2    1    2
## 6  16    4    3          1         2        0      5        4     2    1    2
##   health absences Grades
## 1      3        6      6
## 2      3        4      6
## 3      3       10     10
## 4      5        2     15
## 5      5        4     10
## 6      5       10     15
```

```
n = dim(stu_performance)[1] ## sample size
p = dim(stu_performance)[2] - 1 ## number of non-intercept predictors
```

We start by fitting the **full linear model** including all the predictors:

```
performance.full = lm(Grades ~ ., data=stu_performance)
summary(performance.full)
```

```
##
## Call:
## lm(formula = Grades ~ ., data = stu_performance)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
```

```
## -12.2769  -2.1753   0.4053   2.7927   8.7122
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 13.53702    3.33724   4.056 6.05e-05 ***
## age         -0.25623    0.17880  -1.433  0.15268
## Medu         0.58494    0.25650   2.280  0.02313 *
## Fedu        -0.09135    0.25465  -0.359  0.72000
## traveltime  -0.40184    0.31439  -1.278  0.20197
## studytime    0.26583    0.26903   0.988  0.32373
## failures    -1.83146    0.31201  -5.870 9.49e-09 ***
## famrel       0.28018    0.24537   1.142  0.25422
## freetime     0.34139    0.23096   1.478  0.14021
## goout       -0.62638    0.22179  -2.824  0.00499 **
## Dalc        -0.13723    0.32034  -0.428  0.66862
## Walc         0.35312    0.23834   1.482  0.13928
## health      -0.17429    0.15620  -1.116  0.26521
## absences     0.03197    0.02760   1.158  0.24745
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.211 on 381 degrees of freedom
## Multiple R-squared:  0.183,  Adjusted R-squared:  0.1551
## F-statistic: 6.564 on 13 and 381 DF,  p-value: 2.525e-11
```

We may want to do model selection using the *testing approach* (either via a backward/forward selection) removing or adding one variable at a time, or we can even compare the full model with a reduced containing only the variables that seem to be statistically significant:

```
performance.red = lm(Grades ~ failures + goout + Medu, data=stu_performance)

## Model comparison based on a partial F test
anova(performance.red, performance.full)
```

```
## Analysis of Variance Table
##
## Model 1: Grades ~ failures + goout + Medu
## Model 2: Grades ~ age + Medu + Fedu + traveltime + studytime + failures +
##     famrel + freetime + goout + Dalc + Walc + health + absences
##   Res.Df    RSS Df Sum of Sq     F Pr(>F)
## 1    391 6956.9
## 2    381 6756.7 10    200.22 1.129  0.339
```

In this case, we fail to reject the null so the reduced model is preferred. However, this approach is not efficient nor optimal. Therefore, we use algorithms to search the space of models and a penalty to penalize the inclusion of terms that are

not as useful.

3.4.0.1 Subset Selection via Level-wise algorithms & the `leaps` package

```
library(leaps)
regsubsets_model = regsubsets(Grades~ ., data=stu_performance, nvmax = p)
rs = summary(regsubsets_model)
```

The `rs` object above contains all the different criteria for all the models that have been evaluated. The default maximum size of models is **8** and this is changed above with the `nvmax` option to $p = 13$.

Below we extract the R^2, Adjusted R^2, AIC, BIC and C_p-Mallows criteria for the 13 models that have been evaluated:

```
## Extract the models' R^2
rs$rsq
```

```
##  [1] 0.1298989 0.1483134 0.1587670 0.1633698 0.1668323 0.1696634 0.1724079
##  [8] 0.1746913 0.1773575 0.1800384 0.1823149 0.1827012 0.1829771
```

The best model is the 13th model (as expected).

```
## Extract the models' Adjusted R^2
rs$adjr2
```

```
##  [1] 0.1276849 0.1439681 0.1523125 0.1547890 0.1561232 0.1568231 0.1574385
##  [8] 0.1575865 0.1581269 0.1586853 0.1588305 0.1570269 0.1550997
```

The best model is the 11th model.

```
## Extract the models' Cp Mallows
rs$cp
```

```
##  [1] 14.751803  8.164618  5.289832  5.143431  5.528772  6.208548  6.928720
##  [8]  7.863899  8.620578  9.370388 10.308796 12.128682 14.000000
```

The best model is the 4th model.

```
## Extract the models' BIC
rs$bic
```

```
##  [1] -43.004859 -45.475340 -44.374670 -40.562949 -36.222213 -31.587827
##  [7] -26.916684 -22.029154 -17.328400 -12.638892  -7.758187  -1.965927
## [13]   3.879571
```

The best model is the 2nd model.

If we want to identify which variables are included in each model, then we use the `rs$which` command, where the rows correspond to the model and the columns

to the variable. TRUE means that the variable is included in the model and FALSE means otherwise.

```
rs$which
```

```
##    (Intercept)   age  Medu  Fedu traveltime studytime failures famrel freetime
## 1         TRUE FALSE FALSE FALSE      FALSE     FALSE     TRUE  FALSE    FALSE
## 2         TRUE FALSE  TRUE FALSE      FALSE     FALSE     TRUE  FALSE    FALSE
## 3         TRUE FALSE  TRUE FALSE      FALSE     FALSE     TRUE  FALSE    FALSE
## 4         TRUE FALSE  TRUE FALSE      FALSE     FALSE     TRUE  FALSE     TRUE
## 5         TRUE FALSE  TRUE FALSE       TRUE     FALSE     TRUE  FALSE     TRUE
## 6         TRUE FALSE  TRUE FALSE       TRUE     FALSE     TRUE  FALSE     TRUE
## 7         TRUE  TRUE  TRUE FALSE       TRUE     FALSE     TRUE  FALSE     TRUE
## 8         TRUE  TRUE  TRUE FALSE       TRUE     FALSE     TRUE  FALSE     TRUE
## 9         TRUE  TRUE  TRUE FALSE       TRUE     FALSE     TRUE   TRUE     TRUE
## 10        TRUE  TRUE  TRUE FALSE       TRUE     FALSE     TRUE   TRUE     TRUE
## 11        TRUE  TRUE  TRUE FALSE       TRUE      TRUE     TRUE   TRUE     TRUE
## 12        TRUE  TRUE  TRUE FALSE       TRUE      TRUE     TRUE   TRUE     TRUE
## 13        TRUE  TRUE  TRUE  TRUE       TRUE      TRUE     TRUE   TRUE     TRUE
##    goout  Dalc  Walc health absences
## 1  FALSE FALSE FALSE  FALSE    FALSE
## 2  FALSE FALSE FALSE  FALSE    FALSE
## 3   TRUE FALSE FALSE  FALSE    FALSE
## 4   TRUE FALSE FALSE  FALSE    FALSE
## 5   TRUE FALSE FALSE  FALSE    FALSE
## 6   TRUE FALSE  TRUE  FALSE    FALSE
## 7   TRUE FALSE FALSE  FALSE     TRUE
## 8   TRUE FALSE  TRUE  FALSE     TRUE
## 9   TRUE FALSE  TRUE   TRUE    FALSE
## 10  TRUE FALSE  TRUE   TRUE     TRUE
## 11  TRUE FALSE  TRUE   TRUE     TRUE
## 12  TRUE  TRUE  TRUE   TRUE     TRUE
## 13  TRUE  TRUE  TRUE   TRUE     TRUE
```

For example, Model 1 only contains `failures`, and the `intercept`.

Next, compute "by hand" the AIC/BIC for those p models and find the one that achieves the the smallest score:

```
msize = 1:p;
par(mfrow=c(1,2))
Aic = n*log(rs$rss/n) + 2*msize;
Bic = n*log(rs$rss/n) + msize*log(n);
```

```
Aic
```

```
##  [1] 1148.427 1141.978 1139.100 1138.932 1139.294 1139.950 1140.642 1141.551
##  [9] 1142.272 1142.983 1143.885 1145.698 1147.565
```

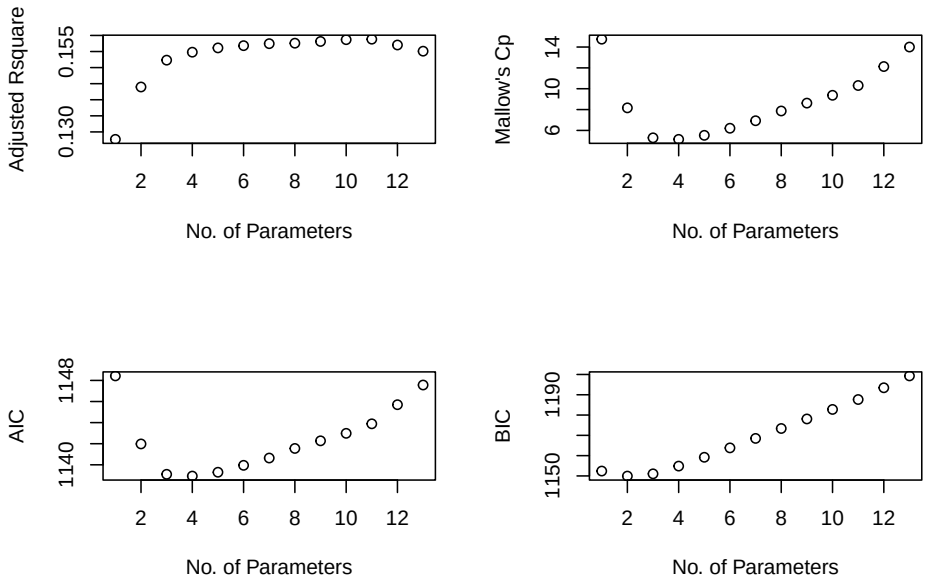
AIC chooses the 4th model.

```
Bic
```

```
##  [1] 1152.406 1149.936 1151.036 1154.848 1159.189 1163.823 1168.494 1173.382
##  [9] 1178.082 1182.772 1187.653 1193.445 1199.290
```

As we saw before, BIC selects the 2nd model. We can also plot the results for the various models. So, we have

```
par(mfrow=c(2,2))
plot(msize, rs$adjr2, xlab="No. of Parameters", ylab = "Adjusted Rsquare");
plot(msize, rs$cp, xlab="No. of Parameters", ylab = "Mallow's Cp");
plot(msize, Aic, xlab="No. of Parameters", ylab = "AIC")
plot(msize, Bic, xlab="No. of Parameters", ylab = "BIC")
```



For this particular data set, AIC and BIC end up selecting different models. **The model selected by AIC is larger than the one selected by BIC**, which is common. AIC favors larger models while BIC favors smaller models. Although the model selected by BIC does not have the smallest AIC score, its AIC score is very close to the smallest one.

```
cbind(rs$which[which.min(Aic),], rs$which[which.min(Bic), ])
```

```
##                [,1]  [,2]
## (Intercept)  TRUE  TRUE
## age         FALSE FALSE
## Medu         TRUE  TRUE
## Fedu        FALSE FALSE
## traveltime  FALSE FALSE
```

```
## studytime   FALSE FALSE
## failures     TRUE  TRUE
## famrel      FALSE FALSE
## freetime     TRUE FALSE
## goout        TRUE FALSE
## Dalc        FALSE FALSE
## Walc        FALSE FALSE
## health      FALSE FALSE
## absences    FALSE FALSE
```

`leaps` does not return AIC, but BIC. Its BIC differs from what has been computed above, but the difference is a constant, so the two BIC formulas (ours and the one used by leaps) are essentially the same.

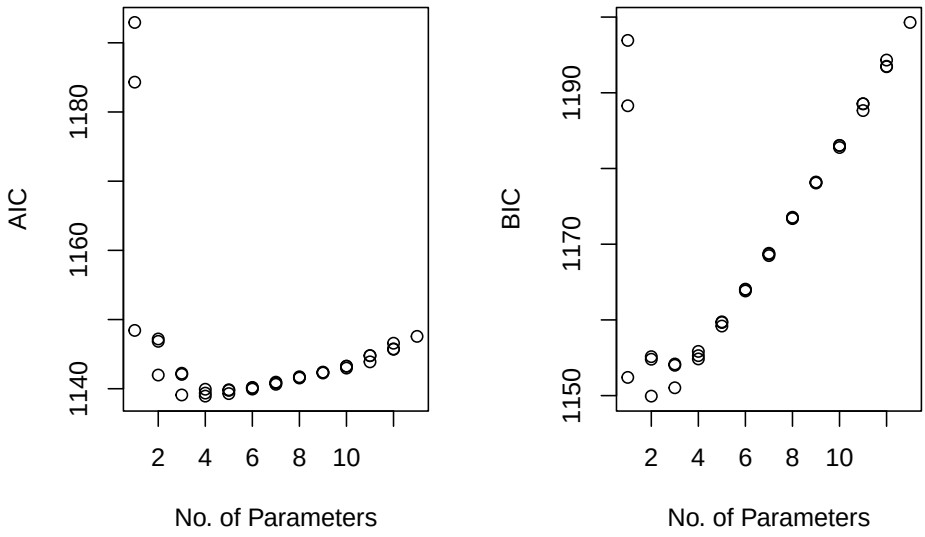
```
cbind(rs$bic, Bic, rs$bic - Bic)
```

```
##                       Bic
##  [1,] -43.004859 1152.406 -1195.411
##  [2,] -45.475340 1149.936 -1195.411
##  [3,] -44.374670 1151.036 -1195.411
##  [4,] -40.562949 1154.848 -1195.411
##  [5,] -36.222213 1159.189 -1195.411
##  [6,] -31.587827 1163.823 -1195.411
##  [7,] -26.916684 1168.494 -1195.411
##  [8,] -22.029154 1173.382 -1195.411
##  [9,] -17.328400 1178.082 -1195.411
## [10,] -12.638892 1182.772 -1195.411
## [11,]  -7.758187 1187.653 -1195.411
## [12,]  -1.965927 1193.445 -1195.411
## [13,]   3.879571 1199.290 -1195.411
```

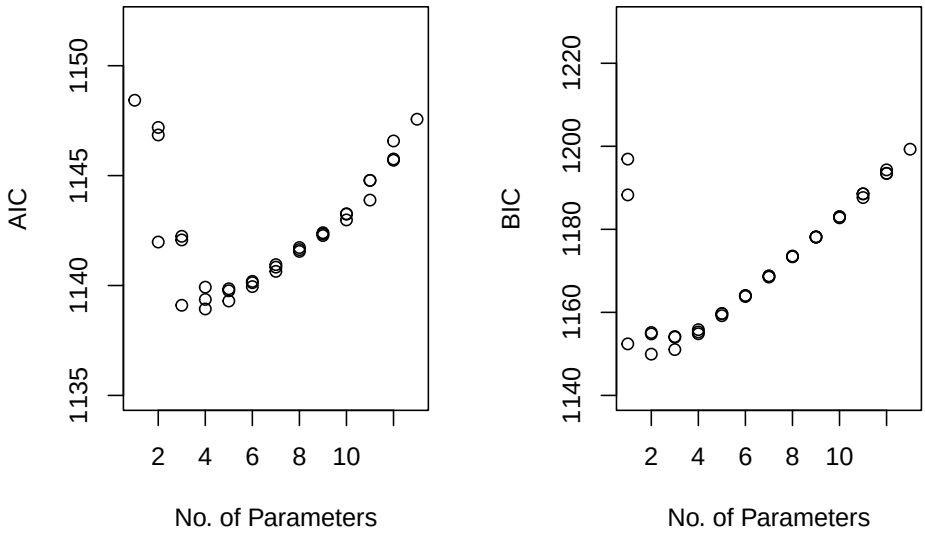
What are the 2nd and 3rd best models in terms of AIC/BIC? This cannot be answered by looking at the AIC/BIC plots shown above. Instead, we need to run the following code.

```
regsubsets_model.1 = regsubsets(Grades ~ ., data=stu_performance, nbest = 3, nvmax = p)
rs.1 = summary(regsubsets_model.1)
#rs.1$which
msize.1 = apply(rs.1$which, 1, sum) - 1
```

```
par(mfrow=c(1,2))
Aic.1 = n*log(rs.1$rss/n) + 2*msize.1
Bic.1 = n*log(rs.1$rss/n) + msize.1*log(n)
plot(msize.1, Aic.1, xlab="No. of Parameters", ylab = "AIC")
plot(msize.1, Bic.1, xlab="No. of Parameters", ylab = "BIC")
```




```
par(mfrow=c(1,2))
plot(msize.1, Aic.1, ylim = c(1135, 1152), xlab="No. of Parameters", ylab = "AIC")
plot(msize.1, Bic.1, ylim = c(1140, 1230), xlab="No. of Parameters", ylab = "BIC")
```




```
# top three models by AIC
rs.1$which[order(Aic.1)[1:3],]
```

```
##   (Intercept)   age Medu  Fedu traveltime studytime failures famrel freetime
## 4        TRUE FALSE TRUE FALSE      FALSE     FALSE     TRUE  FALSE     TRUE
## 3        TRUE FALSE TRUE FALSE      FALSE     FALSE     TRUE  FALSE    FALSE
## 5        TRUE FALSE TRUE FALSE       TRUE     FALSE     TRUE  FALSE     TRUE
##   goout  Dalc  Walc health absences
```

```
## 4  TRUE FALSE FALSE  FALSE    FALSE
## 3  TRUE FALSE FALSE  FALSE    FALSE
## 5  TRUE FALSE FALSE  FALSE    FALSE
```

```
# top three models by BIC
rs.1$which[order(Bic.1)[1:3],]
```

```
##   (Intercept)   age  Medu  Fedu traveltime studytime failures famrel freetime
## 2        TRUE FALSE  TRUE FALSE      FALSE     FALSE     TRUE  FALSE    FALSE
## 3        TRUE FALSE  TRUE FALSE      FALSE     FALSE     TRUE  FALSE    FALSE
## 1        TRUE FALSE FALSE FALSE      FALSE     FALSE     TRUE  FALSE    FALSE
##   goout  Dalc  Walc health absences
## 2 FALSE FALSE FALSE  FALSE    FALSE
## 3  TRUE FALSE FALSE  FALSE    FALSE
## 1 FALSE FALSE FALSE  FALSE    FALSE
```

3.4.0.2 Subset Selection via the step fucntion

We can use the **stepwise** approach to search possible models moving forward, backwards or in both directions (stepwise).

The `R` function to do this is the `step` function. The default is the stepwise method, so if we want to change this, we choose the `direction` option. The method by default outputs **all** the steps of the algorithm, so we can set `trace=0` if we do not want to see the intermediate results. Also, the default criterion for model selection is the `AIC`. However, we can select the BIC criterion by selecting `k = long(n)`.

```
## Stepwise AIC

stepAIC = step(performance.full, trace=0, direction="both")
#stepAIC = step(performance.full, direction="both")
```

```
## Stepwise BIC

n = dim(stu_performance)[1]
stepBIC = step(performance.full, trace=0, direction="both", k=log(n))
```

If we want to retrieve the output from the `step` function, then we can use the following code:

```
sel.var.AIC = attr(stepAIC$terms, "term.labels")
sel.var.BIC = attr(stepBIC$terms, "term.labels")
sel.var.AIC
```

```
## [1] "Medu"     "failures" "freetime" "goout"
```

```
length(sel.var.AIC)
```

```
## [1] 4
```

```
length(sel.var.BIC)
```

```
## [1] 2
```

```
c("Medu", "failures","freetime", "goout") %in% sel.var.AIC
```

```
## [1] TRUE TRUE TRUE TRUE
```

```
sel.var.BIC %in% sel.var.AIC
```

```
## [1] TRUE TRUE
```

3.4.0.3 LASSO and RIDGE Regression

`glmnet` is the library needed for both lasso ($\alpha = 1$) and ridge ($\alpha = 0$). Check more `glmnet` examples at https://web.stanford.edu/~hastie/glmnet/glmnet_beta.html

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-10
```

```
## The input in the glmnet function should be in matrix format:

X = as.matrix(stu_performance[, names(stu_performance) != "Grades"])
Y = stu_performance$Grades
```

We are going to split the data in training (80%) and testing (20%) data sets. Then, we will select the variables in the training set and estimate the coefficients and compute the average MSE on the testing data.

```
## Training/ Testing Data Sets

ntest = round(n*0.2);    ## testing set sample size
ntrain = n - ntest;      ## training set sample size
test.id = sample(1:n, ntest);   ## sampling indices for the test data

Ytest = Y[test.id]; ## Response in the Testing data
```

Training the **full model** in the *training* data set:

```
## Fit the Full model in the training data
full.model = lm(Y[-test.id]~., data = stu_performance[-test.id, ])
```

We then use the full model to **predict** values and *compute the MSE on the testing data set*:

```
## Predicted values on the testing data set
Ytest.pred = predict(full.model, newdata= stu_performance[test.id, ]);
```

```
## Averaged MSE on the test set
sum((Ytest - Ytest.pred)^2)/ntest
```

```
## [1] 1.630006e-29
```

Chapter 4

NonLinear Regression

As we discussed in the previous sections, a Multiple Linear Regression model has a set of assumptions on the error terms (normality, homoscedasticity, independence) as well as limitations, such as the linearity assumption.

If one or more of these assumptions are not satisfied, then the applicability of our model is restricted. In this section, we will introduce models that lift the linearity assumption. Therefore, we are going to discuss the *three* most popular *nonlinear* regression models:

(i) Polynomial Regression
(ii) Splines Regression
(iii) Smoothing Splines

4.0.1 A Note on Nonlinearity

Assume that the true underlying model is of the following form:

$$Y = f(\mathbf{X}) + \varepsilon$$

where ε satisfies the usual assumptions. So far we have been **approximating** the unknown function f via a multiple linear regression model:

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

The MLR model above is **linear both** with respect to the predictors (X_{ij}) **and** with respect to the coefficients (β_j).

Consider now the following models:

$$Y_i = \beta_1 X_{i1} + \beta_2 X_{i1}^2 + \beta_3 X_{i2} + \beta_4 X_{i2}^2 + \beta_5 X_{i1} X_{i2} + \varepsilon_i$$

or

$$\log_{10} Y_i = \beta_1 X_{i1} + \beta_2 \sqrt{X_{i1}} + \beta_3 e^{X_{i3}} + \varepsilon_i$$

These two models are both *non-linear* with respect to the predictors but **linear** with respect to the coefficients β_j.

On the other side, the model below

$$Y_i = \frac{\gamma_0}{1 + \gamma_1 e^{\gamma_2 X_i}} + \varepsilon_i$$

is non-linear both with respect to the parameters *and* the predictors.

During the course of this week, we are going to approximate the general non-linear model, using nonlinear models with respect to the predictors (so that we describe the underlying nonlinear relationship between response and features), but all the models that we will introduce with be *linear with respect to the coefficients/parameters*. In this way, one can think of the models we will work with as models in which the features have been transformed to a new predictor in a nonlinear fashion.

The main advantage of this approach is that we manage to describe (quite efficientily most of the time) a nonlinear relationship between predictor and features, but at the same time we are able to estimate the model parameters easily.

To be more specific, the coefficients will still be obtained via a least-squares approach which (due to the linearity wrt predictor assumption) will result in a system of linear equations as before.

4.1 Polynomial Regression

The simplest form of nonlinear regression is the polynomial regression which is an extension of the linear model by adding higher order terms of the predictor(s). To study this type of regression, we need to first define the polynomial basis functions.

4.1.1 Polynomial Basis Functions

If $b_j(x)$ is the jth basis function, then f has the following representation

$$f(x) = \sum_{j=0}^{d} b_j(x)\beta_j$$

for some values β_j. Therefore, we can write the nonlinear model $y_i = f(x_i) + \varepsilon_i$ as a linear model (with respect to the coefficients)

$$y_i = \beta_0 + \sum_{j=1}^{d} b_j(x_i)\beta_j + \varepsilon_i$$

Suppose that f is believed to be a 4th order polynomial, so *the space of polynomials of order 4 and below contains* f.

A basis for this space is

$$
\begin{aligned}
b_0(x) &= 1 \\
b_1(x) &= x \\
b_2(x) &= x^2 \\
b_3(x) &= x^3 \\
b_4(x) &= x^4
\end{aligned}
$$

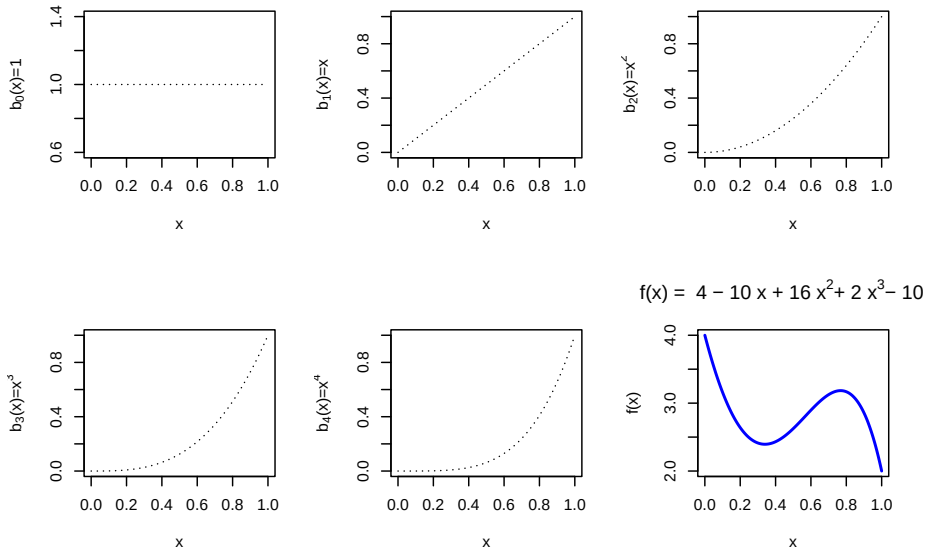
so that the model becomes

$$
y_i = \underbrace{\beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \beta_4 x_i^4}_{=\beta_0 + \sum_{j=1}^{d} b_j(x_i)\beta_j} + \varepsilon_i
$$

Illustration of the Polynomial Basis Functions

Representation of a function in terms of basis functions using a polynomial basis. The following code creates the plots the polynomial basis function up to order 4

```
x=seq(0, 1, by=0.001)
b0 = rep(1, length(x))
b1 = x
b2 = x^2
b3 = x^3
b4 = x^4

fun1 = 4*b0 -10* b1 + 16*b2 + 2*b3 -10*b4

par(mfrow = c(2,3))
plot(x, b0, type='l', lty=3, ylab=expression("b"[0]*"(x)=1"))
plot(x, b1, type='l',lty=3, ylab=expression("b"[1]*"(x)=x"))
plot(x, b2, type='l',lty=3, ylab=expression("b"[2]*"(x)=x"^2))
plot(x, b3, type='l',lty=3, ylab=expression("b"[3]*"(x)=x"^3))
plot(x, b4, type='l',lty=3, ylab=expression("b"[4]*"(x)=x"^4))
plot(x, fun1, type='l', ylab="f(x)", main=expression("f(x) =  4 - 10 x + 16 x"^2*"+ 2 x"^3*"- 10
```



4.1.2 Polynomial Regression

From now on, assume $x \in \mathbb{R}$ is one-dimensional, and extensions to multi-dimensional cases will be discussed later. So, for $x_i \in \mathbb{R}$,

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots + \beta_d x_i^d + \varepsilon_i$$

Then, we create the new variables $X_2 = X^2, \ldots, X_d = X^d$, and treat this as a multiple linear regression model:

$$\underset{n\times 1}{\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}} = \underset{n\times(d+1)}{\begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^d \\ 1 & x_2 & x_2^2 & \cdots & x_2^d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^d \end{pmatrix}} \underset{(d+1)\times 1}{\begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_d \end{pmatrix}} + \varepsilon$$

Therefore, we can say that a polynomial regression model is defined as follows:

Polynomial Regression Model

A *non-linear* model can be represented using a basis of polynomial functions as follows:

$$y_i = f(x_i) + \varepsilon_i \quad \longrightarrow \quad y_i = \beta_0 + \sum_{j=1}^{d} b_j(x_i)\beta_j + \varepsilon_i$$

where d is the degree of the polynomial component.

How do we choose d?

1. *Forward Approach*: Keep *adding* terms until the *last* added term is not significant.

2. *Backward Approach*: Start with a large d, and keep *eliminating* the terms that are not statistically significant, starting with the highest order term.

Once we pick a value of d, then we usually do **not** test the significance of the lower-order terms. Therefore, when we decide to use a polynomial of degree d, by default, we include *all the lower-order terms in our model*.

Reasoning In regression analysis, we do not want our results to be affected by a change of location/scale of the data. Consider the following example: Suppose the data $\{y_i, x_i\}_{i=1}^n$ are generated by the model:

$$y_i = x_i^2 + \varepsilon_i, \quad \varepsilon_i \sim \mathcal{N}(0, \sigma^2)$$

But, they are instead recorded as $\{z_i, x_i\}_{i=1}^n$, where $z_i = x_i + 2$, that is,

$$y_i = (z_i - 2)^2 + \varepsilon_i = 4 - 4z_i + z_i^2 + \varepsilon_i$$

The linear term could become significant, if we shift the x values.

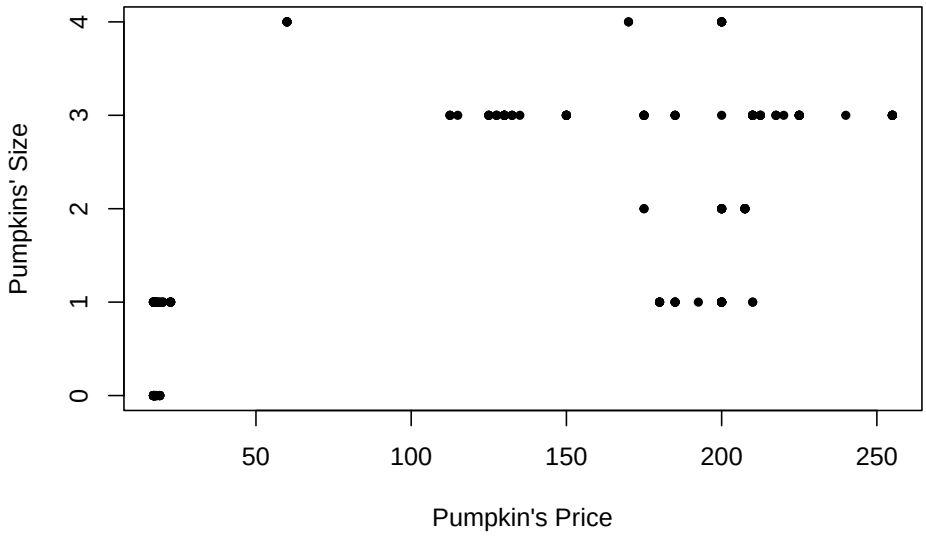
Exception: When we have a particular polynomial function in mind, e.g. the data are collected to test a particular physics formula $Y \approx X^2 + constant$, then you should test whether you can drop the linear term.

The Chicago Pumpkins Example

The `pumpkins.csv` data set contains information regarding the `size` and `price` of pumpkins sold in the Chicago area (data can be found here.). Our goal in this example is to *predict* the `size of the pumpkin` (response) based on its `price` (predictor).

The scatter plot of the data is shown below:

```
pumpkins = read.csv("data/week4/chicagopumpkins.csv",header=TRUE)
plot(pumpkins$price, pumpkins$size, pch=20, xlab="Pumpkin's Price", ylab="Pumpkins' Size")
```



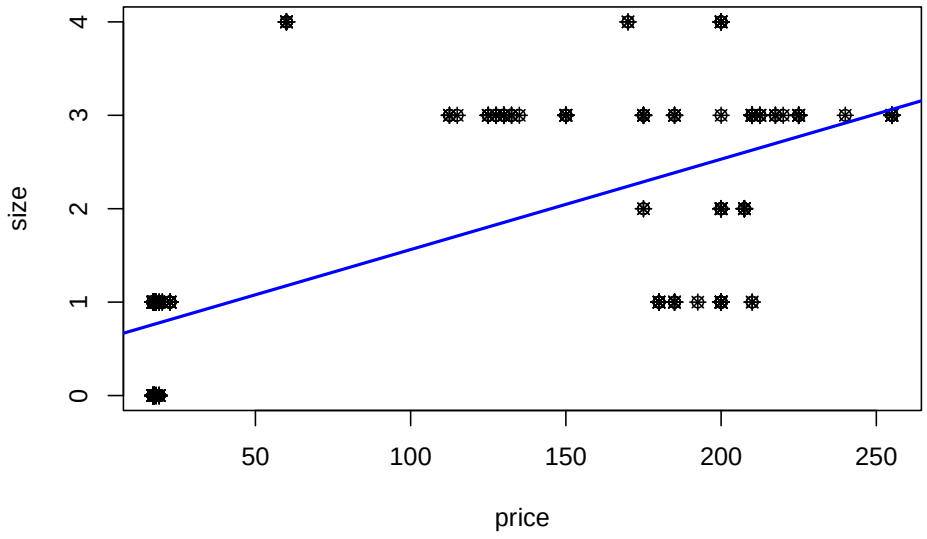
We see that a linear fit is probably *not* a good idea. Indeed,

```
lm.pumpkins = lm(size ~ price, data=pumpkins)
summary(lm.pumpkins)
```

```
##
## Call:
## lm(formula = size ~ price, data = pumpkins)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.6272 -0.7594  0.2244  0.3728  2.8245
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.5948315  0.0988339   6.018 6.35e-09 ***
## price       0.0096781  0.0006856  14.117  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9477 on 246 degrees of freedom
## Multiple R-squared:  0.4475, Adjusted R-squared:  0.4453
## F-statistic: 199.3 on 1 and 246 DF,  p-value: < 2.2e-16
```

Although the predictor is significant at explaining the response, the R^2 is on the lower end and the scatter plot does not support a straight line as a good fit.

```
plot(size ~ price, data=pumpkins)
points(size ~ price, data=pumpkins, pch=8)
abline(lm.pumpkins, col="blue", lwd=2)
```



We want to select a *higher order* model and we will do so following a Forward Selection approach first and a Backward Selection method second:

(1) We start with a Forward Selection approach, i.e. we start by a linear model, and we keep **adding** higher order terms until the added term becomes *statistically insignificant*.

```
# Forward Selection
lm.pumpkins =  lm(size ~ price , data=pumpkins)
summary(lm.pumpkins)
```

```
##
## Call:
## lm(formula = size ~ price, data = pumpkins)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.6272 -0.7594  0.2244  0.3728  2.8245
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.5948315  0.0988339   6.018 6.35e-09 ***
## price       0.0096781  0.0006856  14.117  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9477 on 246 degrees of freedom
## Multiple R-squared:  0.4475, Adjusted R-squared:  0.4453
## F-statistic: 199.3 on 1 and 246 DF,  p-value: < 2.2e-16
```

```
lm.pumpkins2 =  lm(size ~ price + I(price^2), data=pumpkins)
summary(lm.pumpkins2)
```

```
##
## Call:
## lm(formula = size ~ price + I(price^2), data = pumpkins)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.6438 -0.6029  0.3695  0.4895  2.3941
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.093e-01  1.174e-01   0.932    0.353
## price        3.037e-02  3.208e-03   9.469  < 2e-16 ***
## I(price^2)  -9.051e-05  1.375e-05  -6.582  2.8e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 0.8754 on 245 degrees of freedom
## Multiple R-squared:  0.5305, Adjusted R-squared:  0.5267
## F-statistic: 138.4 on 2 and 245 DF,  p-value: < 2.2e-16
```

```
lm.pumpkins3 =  lm(size ~ price + I(price^2) + I(price^3), data=pumpkins)
summary(lm.pumpkins3)
```

```
##
## Call:
## lm(formula = size ~ price + I(price^2) + I(price^3), data = pumpkins)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.2786 -0.4937 -0.1368  0.5531  1.8633
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.414e+00  1.691e-01   -8.36 4.83e-15 ***
## price        1.256e-01  9.079e-03   13.83  < 2e-16 ***
## I(price^2)  -9.845e-04  8.239e-05  -11.95  < 2e-16 ***
## I(price^3)   2.228e-06  2.034e-07   10.95  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7182 on 244 degrees of freedom
## Multiple R-squared:  0.6853, Adjusted R-squared:  0.6814
## F-statistic: 177.1 on 3 and 244 DF,  p-value: < 2.2e-16
```

```
lm.pumpkins4 =  lm(size ~ price + I(price^2) + I(price^3) + I(price^4), data=pumpkins)
summary(lm.pumpkins4)
```

```
##
## Call:
## lm(formula = size ~ price + I(price^2) + I(price^3) + I(price^4),
##     data = pumpkins)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.3200 -0.4497 -0.1241  0.5539  1.7925
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2.871e+00  3.782e-01  -7.590 6.83e-13 ***
## price        2.314e-01  2.630e-02   8.800 2.60e-16 ***
## I(price^2)  -2.565e-03  3.785e-04  -6.776 9.25e-11 ***
## I(price^3)   1.061e-05  1.973e-06   5.378 1.76e-07 ***
```

```
## I(price^4)  -1.470e-08  3.443e-09  -4.271 2.80e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6941 on 243 degrees of freedom
## Multiple R-squared:  0.7073, Adjusted R-squared:  0.7025
## F-statistic: 146.8 on 4 and 243 DF,  p-value: < 2.2e-16
```

```
lm.pumpkins5 =  lm(size ~ price + I(price^2) + I(price^3) + I(price^4)+ I(price^5), data=pumpkins
summary(lm.pumpkins5)
```

```
##
## Call:
## lm(formula = size ~ price + I(price^2) + I(price^3) + I(price^4) +
##     I(price^5), data = pumpkins)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.38082 -0.46537 -0.08211  0.53463  1.91954
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.691e+00  7.112e-01  -2.378   0.0182 *
## price        1.305e-01  5.791e-02   2.253   0.0251 *
## I(price^2)  -2.479e-04  1.244e-03  -0.199   0.8422
## I(price^3)  -1.078e-05  1.113e-05  -0.969   0.3333
## I(price^4)   7.118e-08  4.409e-08   1.615   0.1077
## I(price^5)  -1.248e-10  6.386e-11  -1.954   0.0519 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6901 on 242 degrees of freedom
## Multiple R-squared:  0.7118, Adjusted R-squared:  0.7059
## F-statistic: 119.6 on 5 and 242 DF,  p-value: < 2.2e-16
```

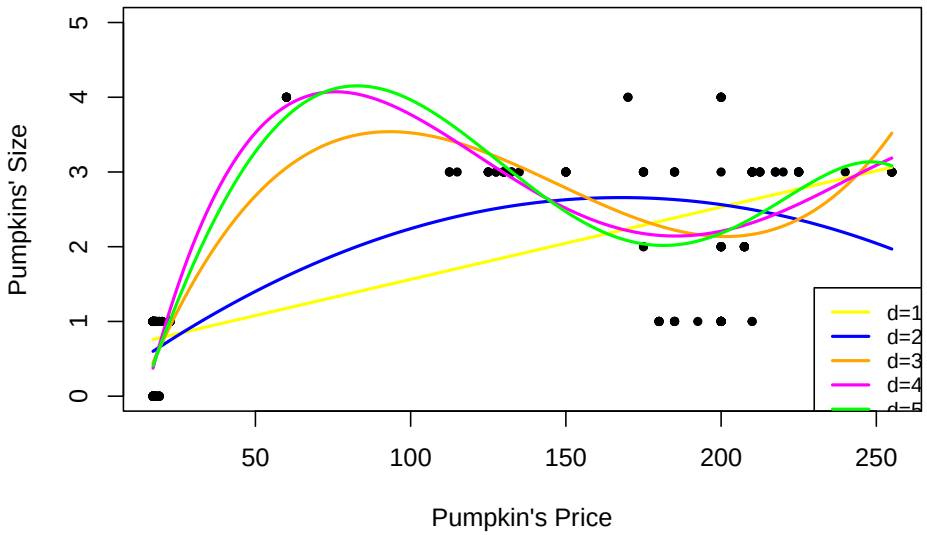
We see that the *5th order model* has a 5th order term with *p-value equal to 0.0519*, which is **higher than 5%**, so we conclude that the optimal order for the polynomial, according to the forward selection method is $d = 4$. So, the fitted model is:

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x + \hat{\beta}_2 x^2 + \hat{\beta}_3 x^4 + \hat{\beta}_4 x^4$$

If we plot all the fitted models, we have:

```
newprice = data.frame(price=seq(17, 255, 1))
plot(pumpkins$price, pumpkins$size, pch=20, ylim=c(0,5), xlab="Pumpkin's Price", ylab="Pumpkins'
lines(newprice$price, predict(lm.pumpkins, newprice), col="yellow", lty=1, lwd=2);
lines(newprice$price, predict(lm.pumpkins2, newprice), col="blue", lty=1, lwd=2);
```

```
lines(newprice$price, predict(lm.pumpkins3, newprice), col="orange", lty=1, lwd=2);
lines(newprice$price, predict(lm.pumpkins4, newprice), col="magenta", lty=1, lwd=2);
lines(newprice$price, predict(lm.pumpkins5, newprice), col="green", lty=1, lwd=2);
legend(230, 1.45, legend=c("d=1", "d=2", "d=3", "d=4", "d=5"),
       col=c("yellow", "blue", "orange", "magenta", "green"), lty=c(1,1,1,1,1), cex=0.8
```



The magenta line is the one that corresponds to the 4th order model.

(2) We can also select d using the Backward Elimination approach, that is we start with a large value for d and we eliminate terms *until the highest order term in the model is statistically significant*:

```
lm.pumpkins10 =  lm(size ~ price + I(price^2) + I(price^3) + I(price^4)+ I(price^5)+ I
summary(lm.pumpkins10)
```

```
##
## Call:
## lm(formula = size ~ price + I(price^2) + I(price^3) + I(price^4) +
##     I(price^5) + I(price^6) + I(price^7) + I(price^8) + I(price^9) +
##     I(price^10), data = pumpkins)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.44082 -0.45530 -0.01853  0.53535  2.12546
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.331e+01  2.671e+01   0.498    0.619
```

```
## price       -2.191e+00  4.274e+00  -0.513    0.609
## I(price^2)   1.396e-01  2.625e-01   0.532    0.595
## I(price^3)  -4.389e-03  8.140e-03  -0.539    0.590
## I(price^4)   8.198e-05  1.458e-04   0.562    0.575
## I(price^5)  -9.818e-07  1.624e-06  -0.605    0.546
## I(price^6)   7.731e-09  1.163e-08   0.664    0.507
## I(price^7)  -3.970e-11  5.374e-11  -0.739    0.461
## I(price^8)   1.276e-13  1.549e-13   0.824    0.411
## I(price^9)  -2.321e-16  2.535e-16  -0.916    0.361
## I(price^10)  1.821e-19  1.800e-19   1.012    0.313
##
## Residual standard error: 0.6483 on 237 degrees of freedom
## Multiple R-squared:  0.7509, Adjusted R-squared:  0.7404
## F-statistic: 71.45 on 10 and 237 DF,  p-value: < 2.2e-16
```

```
lm.pumpkins9 =  lm(size ~ price + I(price^2) + I(price^3) + I(price^4)+ I(price^5)+ I(price^6) +
summary(lm.pumpkins9)
```

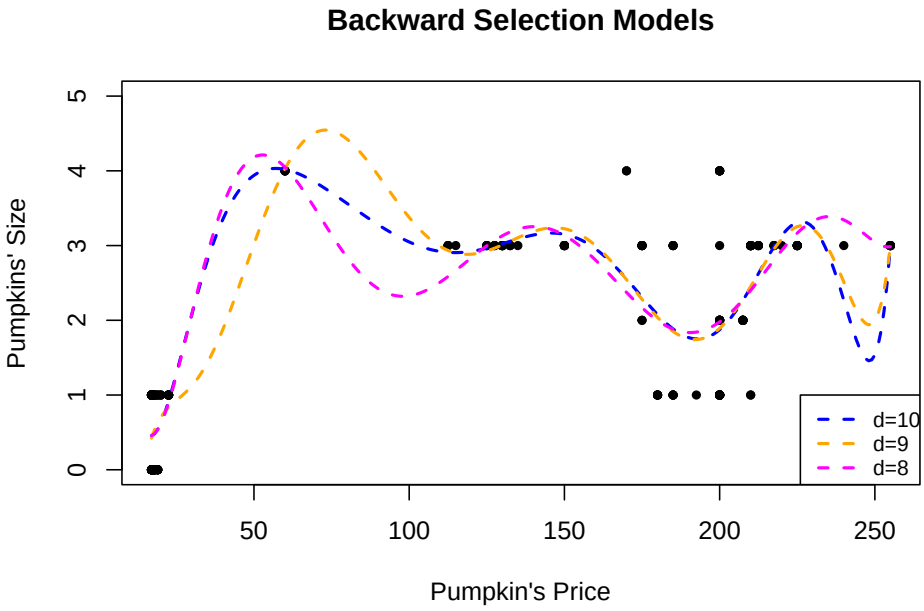
```
##
## Call:
## lm(formula = size ~ price + I(price^2) + I(price^3) + I(price^4) +
##     I(price^5) + I(price^6) + I(price^7) + I(price^8) + I(price^9),
##     data = pumpkins)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.4514 -0.4230 -0.0091  0.5177  2.1072
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.191e+01  9.589e+00  -1.242   0.2154
## price        1.877e+00  1.450e+00   1.295   0.1967
## I(price^2)  -1.125e-01  8.271e-02  -1.360   0.1751
## I(price^3)   3.506e-03  2.316e-03   1.514   0.1314
## I(price^4)  -6.094e-05  3.623e-05  -1.682   0.0939 .
## I(price^5)   6.252e-07  3.391e-07   1.844   0.0664 .
## I(price^6)  -3.875e-09  1.945e-09  -1.993   0.0475 *
## I(price^7)   1.425e-11  6.704e-12   2.125   0.0346 *
## I(price^8)  -2.859e-14  1.276e-14  -2.241   0.0259 *
## I(price^9)   2.411e-17  1.030e-17   2.340   0.0201 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6483 on 238 degrees of freedom
## Multiple R-squared:  0.7499, Adjusted R-squared:  0.7404
## F-statistic: 79.27 on 9 and 238 DF,  p-value: < 2.2e-16
```

```
lm.pumpkins8 =  lm(size ~ price + I(price^2) + I(price^3)+ I(price^4)+ I(price^5)+ I(pr
summary(lm.pumpkins8)
```

```
##
## Call:
## lm(formula = size ~ price + I(price^2) + I(price^3) + I(price^4) +
##     I(price^5) + I(price^6) + I(price^7) + I(price^8), data = pumpkins)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.39988 -0.45678 -0.05827  0.53309  2.02119
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  9.140e+00  3.347e+00   2.731 0.006791 **
## price       -1.368e+00  4.257e-01  -3.215 0.001486 **
## I(price^2)   7.522e-02  2.032e-02   3.702 0.000266 ***
## I(price^3)  -1.798e-03  4.783e-04  -3.759 0.000214 ***
## I(price^4)   2.262e-05  6.154e-06   3.675 0.000293 ***
## I(price^5)  -1.611e-07  4.541e-08  -3.549 0.000466 ***
## I(price^6)   6.535e-10  1.918e-10   3.407 0.000771 ***
## I(price^7)  -1.406e-12  4.316e-13  -3.259 0.001282 **
## I(price^8)   1.246e-15  4.008e-16   3.108 0.002112 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6544 on 239 degrees of freedom
## Multiple R-squared:  0.7441, Adjusted R-squared:  0.7355
## F-statistic: 86.87 on 8 and 239 DF,  p-value: < 2.2e-16
```

Starting with an order *10* model, we identify that an *9th* order model is *optimal* according to the backward elimination criterion. If we plot all the fitted models, we have:

```
newprice = data.frame(price=seq(17, 255, 1))
plot(pumpkins$price, pumpkins$size, ylim=c(0,5), pch=20, xlab="Pumpkin's Price", ylab="
lines(newprice$price, predict(lm.pumpkins10, newprice), col="blue", lty=2, lwd=2);
lines(newprice$price, predict(lm.pumpkins9, newprice), col="orange", lty=2, lwd=2);
lines(newprice$price, predict(lm.pumpkins8, newprice), col="magenta", lty=2, lwd=2);
legend(226, 1, legend=c("d=10", "d=9", "d=8"),
       col=c("blue", "orange", "magenta"), lty=c(2,2,2), cex=0.8, lwd=2)
```



The magenta line is the one that corresponds to the *9th* order model.

For the fitted model we finally choose, we should (as always) perform diagnostic tests.

4.1.3 Orthogonal Polynomials

Fitting high order polynomials is generally **not recommended**, since they are very *unstable* and *difficult to interpret*. In addition, successive predictors x^j are *highly correlated* introducing multicollinearity problems. One way around this is to fit **orthogonal polynomials** of the form:

$$y_i = \beta_0 + \beta_1 z_1 + ... + \beta_d z_d + \varepsilon_i$$

where each $z_j = a_1 + b_2 x + ... + \kappa_j x^j$ is a polynomial of order j with coefficients chosen such that $z_i^\top z_j = 0$ (i.e. the inner product of any two polynomials is zero).

The Chicago Pumpkins Example

In `R`, we can fit orthogonal polynomials using the `poly` function. In the code below, we repeat the same process as before (for choosing d) using the orthogonal polynomials.

```
# Forward Selection
lm.pumpkins02 =  lm(size ~ poly(price,2), data=pumpkins)
summary(lm.pumpkins02)
```

```
##
```

```
## Call:
## lm(formula = size ~ poly(price, 2), data = pumpkins)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.6438 -0.6029  0.3695  0.4895  2.3941
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)      1.70161    0.05559  30.612  < 2e-16 ***
## poly(price, 2)1 13.37846    0.87538  15.283  < 2e-16 ***
## poly(price, 2)2 -5.76139    0.87538  -6.582  2.8e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8754 on 245 degrees of freedom
## Multiple R-squared:  0.5305, Adjusted R-squared:  0.5267
## F-statistic: 138.4 on 2 and 245 DF,  p-value: < 2.2e-16
```

```
lm.pumpkins03 =  lm(size ~ poly(price,3), data=pumpkins)
summary(lm.pumpkins03)
```

```
##
## Call:
## lm(formula = size ~ poly(price, 3), data = pumpkins)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.2786 -0.4937 -0.1368  0.5531  1.8633
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)       1.7016     0.0456  37.313  < 2e-16 ***
## poly(price, 3)1  13.3785     0.7182  18.628  < 2e-16 ***
## poly(price, 3)2  -5.7614     0.7182  -8.022 4.35e-14 ***
## poly(price, 3)3   7.8672     0.7182  10.954  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7182 on 244 degrees of freedom
## Multiple R-squared:  0.6853, Adjusted R-squared:  0.6814
## F-statistic: 177.1 on 3 and 244 DF,  p-value: < 2.2e-16
```

```
lm.pumpkins04 =  lm(size ~ poly(price,4), data=pumpkins)
summary(lm.pumpkins04)
```

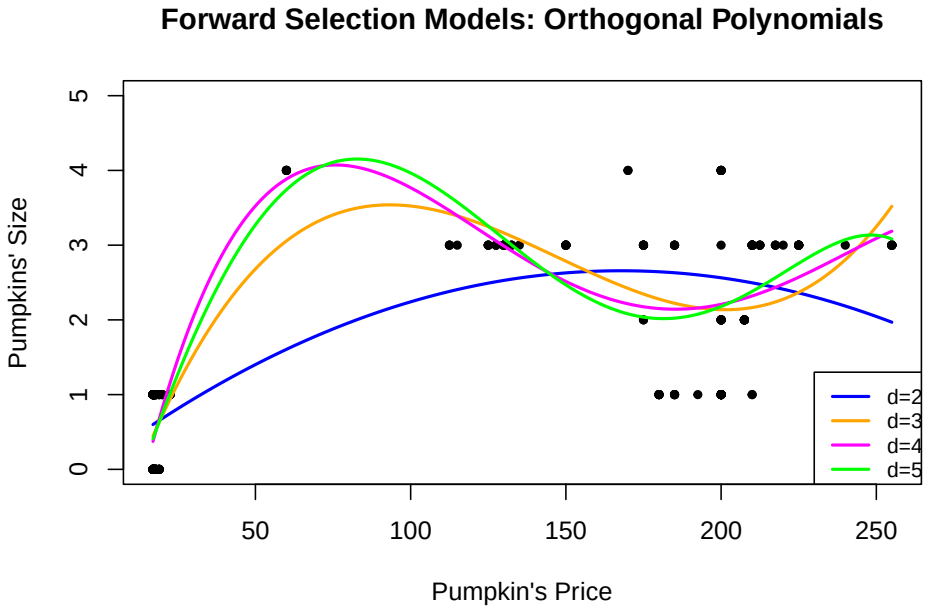
```
##
```

```
## Call:
## lm(formula = size ~ poly(price, 4), data = pumpkins)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.3200 -0.4497 -0.1241  0.5539  1.7925
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)      1.70161    0.04407  38.608  < 2e-16 ***
## poly(price, 4)1 13.37846    0.69408  19.275  < 2e-16 ***
## poly(price, 4)2 -5.76139    0.69408  -8.301 7.22e-15 ***
## poly(price, 4)3  7.86718    0.69408  11.335  < 2e-16 ***
## poly(price, 4)4 -2.96423    0.69408  -4.271 2.80e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6941 on 243 degrees of freedom
## Multiple R-squared:  0.7073, Adjusted R-squared:  0.7025
## F-statistic: 146.8 on 4 and 243 DF,  p-value: < 2.2e-16
```

```
lm.pumpkins05 =  lm(size ~ poly(price,5), data=pumpkins)
summary(lm.pumpkins05)
```

```
##
## Call:
## lm(formula = size ~ poly(price, 5), data = pumpkins)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.38082 -0.46537 -0.08211  0.53463  1.91954
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)      1.70161    0.04382  38.831  < 2e-16 ***
## poly(price, 5)1 13.37846    0.69009  19.387  < 2e-16 ***
## poly(price, 5)2 -5.76139    0.69009  -8.349 5.35e-15 ***
## poly(price, 5)3  7.86718    0.69009  11.400  < 2e-16 ***
## poly(price, 5)4 -2.96423    0.69009  -4.295 2.53e-05 ***
## poly(price, 5)5 -1.34841    0.69009  -1.954   0.0519 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6901 on 242 degrees of freedom
## Multiple R-squared:  0.7118, Adjusted R-squared:  0.7059
## F-statistic: 119.6 on 5 and 242 DF,  p-value: < 2.2e-16
```

```
newprice = data.frame(price=seq(17, 255, 1))
plot(pumpkins$price, pumpkins$size, pch=20,  ylim=c(0,5),xlab="Pumpkin's Price", ylab="
lines(newprice$price, predict(lm.pumpkins02, newprice), col="blue", lty=1, lwd=2);
lines(newprice$price, predict(lm.pumpkins03, newprice), col="orange", lty=1, lwd=2);
lines(newprice$price, predict(lm.pumpkins04, newprice), col="magenta", lty=1, lwd=2);
lines(newprice$price, predict(lm.pumpkins05, newprice), col="green", lty=1, lwd=2);
legend(230, 1.3, legend=c("d=2", "d=3", "d=4", "d=5"),
       col=c("blue", "orange", "magenta", "green"), lty=c(1,1,1,1), cex=0.8, lwd=c(2,2
```




```
# Backward Selection
lm.pumpkins010 =  lm(size ~ poly(price,10), data=pumpkins)
summary(lm.pumpkins010)
```

```
##
## Call:
## lm(formula = size ~ poly(price, 10), data = pumpkins)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.44082 -0.45530 -0.01853  0.53535  2.12546
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)          1.70161    0.04117  41.335  < 2e-16 ***
## poly(price, 10)1    13.37846    0.64829  20.636  < 2e-16 ***
## poly(price, 10)2    -5.76139    0.64829  -8.887  < 2e-16 ***
```

```
## poly(price, 10)3   7.86718    0.64829  12.135  < 2e-16 ***
## poly(price, 10)4  -2.96423    0.64829  -4.572 7.77e-06 ***
## poly(price, 10)5  -1.34841    0.64829  -2.080  0.03861 *
## poly(price, 10)6  -1.45456    0.64829  -2.244  0.02578 *
## poly(price, 10)7  -2.57955    0.64829  -3.979 9.20e-05 ***
## poly(price, 10)8   2.03378    0.64829   3.137  0.00192 **
## poly(price, 10)9   1.51698    0.64829   2.340  0.02012 *
## poly(price, 10)10  0.65591    0.64829   1.012  0.31269
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6483 on 237 degrees of freedom
## Multiple R-squared:  0.7509, Adjusted R-squared:  0.7404
## F-statistic: 71.45 on 10 and 237 DF,  p-value: < 2.2e-16
```

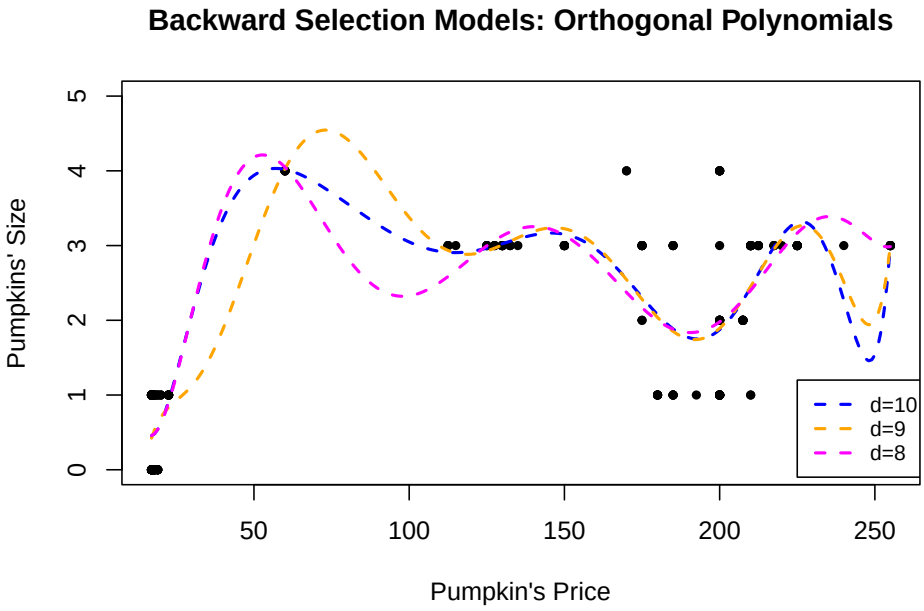
```
lm.pumpkins09 =  lm(size ~ poly(price,9), data=pumpkins)
summary(lm.pumpkins09)
```

```
##
## Call:
## lm(formula = size ~ poly(price, 9), data = pumpkins)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.4514 -0.4230 -0.0091  0.5177  2.1072
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)      1.70161    0.04117  41.333  < 2e-16 ***
## poly(price, 9)1 13.37846    0.64833  20.635  < 2e-16 ***
## poly(price, 9)2 -5.76139    0.64833  -8.887  < 2e-16 ***
## poly(price, 9)3  7.86718    0.64833  12.135  < 2e-16 ***
## poly(price, 9)4 -2.96423    0.64833  -4.572 7.76e-06 ***
## poly(price, 9)5 -1.34841    0.64833  -2.080  0.03861 *
## poly(price, 9)6 -1.45456    0.64833  -2.244  0.02578 *
## poly(price, 9)7 -2.57955    0.64833  -3.979 9.20e-05 ***
## poly(price, 9)8  2.03378    0.64833   3.137  0.00192 **
## poly(price, 9)9  1.51698    0.64833   2.340  0.02012 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6483 on 238 degrees of freedom
## Multiple R-squared:  0.7499, Adjusted R-squared:  0.7404
## F-statistic: 79.27 on 9 and 238 DF,  p-value: < 2.2e-16
```

```
lm.pumpkins08 =  lm(size ~ poly(price,8), data=pumpkins)
summary(lm.pumpkins08)
```

```
##
## Call:
## lm(formula = size ~ poly(price, 8), data = pumpkins)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.39988 -0.45678 -0.05827  0.53309  2.02119
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)       1.70161    0.04155  40.951  < 2e-16 ***
## poly(price, 8)1 13.37846    0.65437  20.445  < 2e-16 ***
## poly(price, 8)2 -5.76139    0.65437  -8.805 2.71e-16 ***
## poly(price, 8)3  7.86718    0.65437  12.023  < 2e-16 ***
## poly(price, 8)4 -2.96423    0.65437  -4.530 9.32e-06 ***
## poly(price, 8)5 -1.34841    0.65437  -2.061 0.040421 *
## poly(price, 8)6 -1.45456    0.65437  -2.223 0.027162 *
## poly(price, 8)7 -2.57955    0.65437  -3.942 0.000106 ***
## poly(price, 8)8  2.03378    0.65437   3.108 0.002112 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6544 on 239 degrees of freedom
## Multiple R-squared:  0.7441, Adjusted R-squared:  0.7355
## F-statistic: 86.87 on 8 and 239 DF,  p-value: < 2.2e-16
```

```
plot(pumpkins$price, pumpkins$size, pch=20, ylim=c(0,5), xlab="Pumpkin's Price", ylab="
lines(newprice$price, predict(lm.pumpkins010, newprice), col="blue", lty=2, lwd=2);
lines(newprice$price, predict(lm.pumpkins09, newprice), col="orange", lty=2, lwd=2);
lines(newprice$price, predict(lm.pumpkins08, newprice), col="magenta", lty=2, lwd=2);
legend(225, 1.2, legend=c("d=10", "d=9", "d=8"),  col=c("blue", "orange", "magenta"), l
```



4.1.4 Piece-wise Polynomials

If the true mean of $\mathbb{E}(Y|X = x) = f(x)$ is *too wiggly*, we might need to fit a higher order polynomial, which is not always a good idea. Instead we consider **piece-wise polynomials**:

1. we divide the range of x into several intervals, and
2. within each interval, $f(x)$ is a low-order polynomial, e.g., cubic or quadratic, but the polynomial coefficients will be different from interval to interval
3. we require the overall $f(x)$ to be continuous up to certain derivatives.

This method is also called *"broken-stick regression"*. Its benefit is that it localizes the influence of each data point to a particular segment, but overall it is not a very smooth line as the one we obtain by fitting a single polynomial for the whole data set.

4.2 Splines Regression

Splines are piecewise polynomials thar are constructed so that thay can be both sensitive and smooth, but also capture local features of the data.

A **Cubic Spline** is a curve constructed from sections of cubic polynomials, joined together so that the curve is *continuous up to second derivative*. The points at which the sections join are called the **knots** of the spline. For a conventional spline, the knots occur wherever there is a datum, but for the

regression splines the locations of the knots must be chosen. Typically, the knots would either be *evenly spaced* through the range of observed x values, or placed at the *quantiles* of the distribution of unique x values. Each section of cubic has different coefficients, but at the knots *it will match its neighboring sections in value and first two derivatives.*

Cubic Splines (Mathematical) Definition

A function g defined on $[a, b]$ is a **cubic spline** with respect to **knots** $\{\xi_i\}_{i=1}^m$ (specifically $a < \xi_1 < \xi_2 < ... < \xi_m < b$) if:

- g is a cubic polynomial in each of the $m + 1$ intervals, i.e.

$$g(x) = d_i x^3 + c_i x^2 + b_i x + a_i, \quad x \in [\xi_i, \xi_{i+1}]$$

  where $i = 0, ..., m$, $\xi_0 = a$ and $\xi_{m+1} = b$.

- g is continuous up to the *2nd* derivative. Since g is continuous up to the *2nd* derivative *for any point inside an interval*, it suffices to check the following conditions:

$$g^{(0,1,2)}(\xi_i^+) = g^{(0,1,2)}(\xi_i^-), \quad i = 1, ..., m$$

  This expression indicates that the function and the first and second order derivatives are continuous at the knots.

How many free parameters do we need to represent a cubic spline?

(+) 4 parameters (d_i, c_i, b_i, a_i) *for each of the* $(m + 1)$ *intervals.*

(-) 3 constraints *at each of the* m *knots* (continuity constraints).

The total number of free parameters (similar to the number of degrees of freedom) is:

$$4(m + 1) - 3m = m + 4$$

A property of the cubic splines

Suppose the knots $\{\xi_i\}_{i=1}^m$ are given.

- If $g_1(x)$ and $g_2(x)$ are cubic splines, the linear combination

$$a_1 g_1(x) + a_2 g_2(x)$$

  is also a cubic spline, where a_1 and a_2 are known constants.

That is, for a set of given knots, the corresponding *cubic splines form a linear space (of functions) with dim* $(m + 4)$.

4.2.1 Examples of Cubic Splines Basis

1. A set of basis functions for cubic splines (w.r.t knots $\{\xi_i\}_{i=1}^m$) is given by:

$$\begin{aligned} h_0(x) &= 1 \\ h_1(x) &= x \\ h_2(x) &= x^2 \\ h_3(x) &= x^3; \\ h_{i+3}(x) &= (x-\xi_i)_+^3, \quad i = 1, 2, \ldots, m \end{aligned}$$

   That is, any cubic spline can be uniquely expressed as:

$$\beta_0 + \sum_{j=1}^{m+3} \beta_j h_j(x)$$

2. Given knot locations, there are many alternative, but equivalent ways of writing down a basis for cubic splines. For example, *another* basis for cubic splines can be written as:

$$\begin{aligned} h_0(x) &= 1 \\ h_1(x) &= x \\ h_{i+1}(x) &= R(x, \xi_i^*), \ i = 1, \ldots, q-1 \end{aligned}$$

   where

$$\begin{aligned} R(x,z) = &\left[(z-1/2)^2 - 1/12\right]\left[(x-1/2)^2 - 1/12\right]/4 \\ &- \left[(|x-z|-1/2)^4 - 1/2(|x-z|-1/2)^2 + 7/240\right]/24 \end{aligned}$$

An Example of a Cubic Splines Basis

We first define the function $R(x, z)$ as above in R:

```
R_xz <- function(x,z){
  ((z-0.5)^2-1/12)*((x-0.5)^2-1/12)/4-((abs(x-z)-0.5)^4-(abs(x-z)-0.5)^2/2+7/240)/24
}
```

We then need to define the knots

```
new.knots= c(1/6, 3/6, 5/6)
```

In regression examples the x variable will be the predictor (there is no need to generate any x values). Here, we need a "generic" x for illustration purposes.

```
x=seq(0, 1, by=0.01)
```

Finally, we defined the splines functions, based on the definition given above:

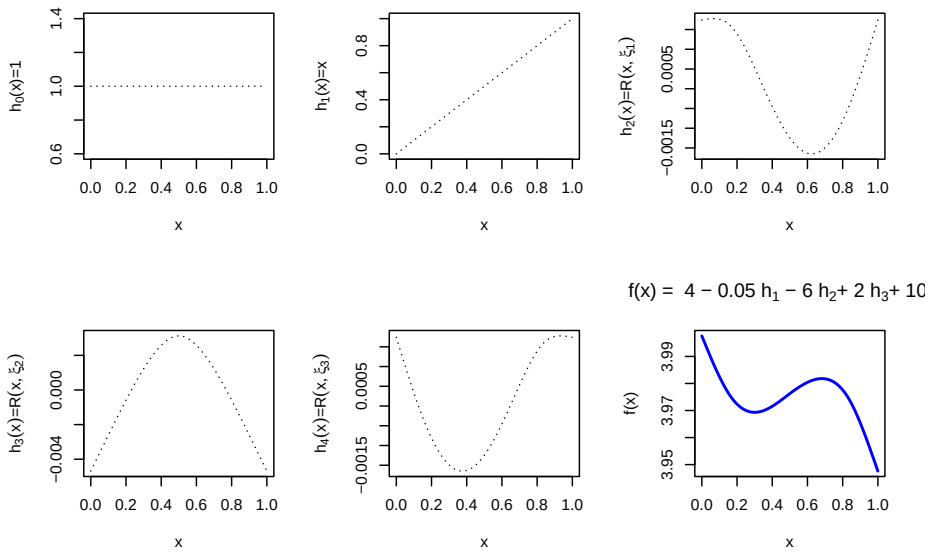
```
spline1 = rep(1, length(x)) # First basis function
spline2 = x # Second basis function
spline3 = R_xz(x,new.knots[1]) # Third basis function
spline4 = R_xz(x,new.knots[2]) # Fourth basis function
spline5 = R_xz(x,new.knots[3]) # Fifth basis function
```

If we want to represent a function using the basis above, we have:

```
fun1s = 4*spline1 - 0.05*spline2 - 6*spline3 +2*spline4 +10*spline5
```

For illustration purposes, we plot the basis functions defined above as well as the function f:

```
par(mfrow = c(2,3))
plot(x, spline1, type='l', lty=3, ylab=expression("h"[0]*"(x)=1"))
plot(x, spline2, type='l',lty=3, ylab=expression("h"[1]*"(x)=x"))
plot(x, spline3, type='l',lty=3, ylab=expression(paste(h[2](x))*"="*paste(R(x,xi[1]))))
plot(x, spline4, type='l',lty=3, ylab=expression(paste(h[3](x))*"="*paste(R(x,xi[2]))))
plot(x, spline5, type='l',lty=3, ylab=expression(paste(h[4](x))*"="*paste(R(x,xi[3]))))
plot(x, fun1s, type='l', ylab="f(x)", main=expression("f(x) =  4 - 0.05 h"[1]*" - 6 h"
```



4.2.2 B-Splines Basis Functions in R

In this class, one of the cubic splines basis that we are going to use is called **B-Splines**.

Cubic B Splines Definition

The cubic B Splines basis is defined on an interval $[a, b]$ by the following requirements on the interior basis functions with knotpoints at $\{\xi_i\}_{i=1}^m$:

1. A given basis function is nonzero on an interval defined by *four successive knots and zero elsewhere.*
2. The basis function is a cubic polynomial for each subinterval between successive knots.
3. The basis function is continuous and is also continuous in its first and second derivatives at each knotpoint.
4. The basis function integrates to 1 over its support.
5. The boundary function definitions are adjusted to account for continuity in derivatives at the boundaries of the interval.

These are part of the `splines` library in `R` and the function to generate then is called `bs`. The arguments of the `bs` function are:

$$bs(x, df, knots, degree, intercept, Boundary.knots)$$

The `R` documentation can be found here.

Arguments in the `bs` function:

- `x` is the predictor, i.e. the variable to which we want to apply the splines function.
- `df` are the degrees of freedom for the splines functions.
- `knots` are the *internal* breakpoints/knots that define the spline, i.e. location where the knots should be placed.
- `degree` is the degree of the piecewise polynomial. The default is `degree=3` for cubic splines.
- `intercept` is whether we want to add an intercept in the splines basis or not. *This is not the intercept in the regression we perform.* The default is `intercept=FALSE`.
- `Boundary.knots` are the boundary points at which we "anchor" the B-splines basis. The default values are obtained from the range of non-NA values of `x`, i.e. `Boundary.knots=FALSE`.

So, *in our examples*, we will need to feed the appropriate predictor `x` and then the desired `knots` **or** degrees of freedom, `df`, *not both.* The rest we leave them at their default values.

The **output** of the `bs()` function is a *matrix* of dimension `c(length(x), df)`, if `df` was supplied or of dimension `df = length(knots) + degree (+1 if intercept=TRUE)` if `knots` were supplied.

Knots or Degrees of Freedom in `bs()`

As we mentioned, we need to supply the degrees of freedom or the location of the knots in the `bs()` function, and of course there is an equivalence between

the two. Some details to keep in mind on how you can correctly specify these arguments:

- If you know where the knots should be placed, then you need to define a vector of the **locations** of the knots and input that in the `knots` argument in the `bs()` function. For example,

Defining `knots` location in `bs()`

```
library(splines)
x=seq(0, 1, by=0.01) # generic `x` for illustration purposes
new.knots= c(1/6, 3/6, 5/6)  # define three knots at locations 1/6, 3/6, 5/6.
Bsplines.basis1 = bs(x, knots=new.knots)
head(Bsplines.basis1)
```

```
##              1         2         3 4 5 6
## [1,] 0.000000 0.0000000 0.0000000 0 0 0
## [2,] 0.165912 0.0034896 0.0000144 0 0 0
## [3,] 0.304896 0.0135168 0.0001152 0 0 0
## [4,] 0.418824 0.0294192 0.0003888 0 0 0
## [5,] 0.509568 0.0505344 0.0009216 0 0 0
## [6,] 0.579000 0.0762000 0.0018000 0 0 0
```

In this case, `R` calculates the degrees of freedom using the following formula:

$$\text{df} = \text{length(knots)} + \text{degree}$$

or **if `intercept=TRUE`** then

$$\text{df} = \text{length(knots)} + \text{degree} + 1$$

- If you prefer to specify the degrees of freedom, then the function `bs()` chooses `df-degree` many knots at suitable quantiles of x (ignoring any missing values). If you specified `intercept=TRUE`, then the number of knots will be `df-degree-1`. For example,

Defining `knots` location in `bs()`

```
x=seq(0, 1, by=0.01) # generic `x` for illustration purposes
Bsplines.basis2 = bs(x, df=4)
head(Bsplines.basis2)
```

```
##              1        2        3 4
## [1,] 0.000000 0.000000 0.000000 0
## [2,] 0.058214 0.000592 0.000002 0
## [3,] 0.112912 0.002336 0.000016 0
## [4,] 0.164178 0.005184 0.000054 0
## [5,] 0.212096 0.009088 0.000128 0
## [6,] 0.256750 0.014000 0.000250 0
```

As an example, below we plot the first 7 B Splines basis functions:

Illustration of B Splines Basis Function

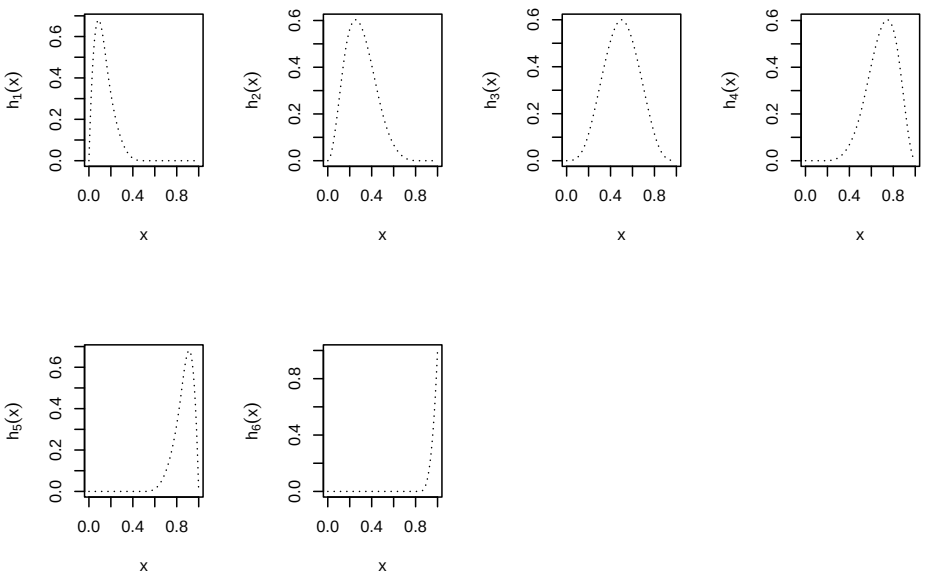
```
Bsplines.basis = bs(x, knots=new.knots)
dim(Bsplines.basis)
```

```
## [1] 101   6
```

```
head(Bsplines.basis)
```

```
##              1         2         3 4 5 6
## [1,] 0.000000 0.0000000 0.0000000 0 0 0
## [2,] 0.165912 0.0034896 0.0000144 0 0 0
## [3,] 0.304896 0.0135168 0.0001152 0 0 0
## [4,] 0.418824 0.0294192 0.0003888 0 0 0
## [5,] 0.509568 0.0505344 0.0009216 0 0 0
## [6,] 0.579000 0.0762000 0.0018000 0 0 0
```

```
par(mfrow = c(2,4))
plot(x, Bsplines.basis[,1], type='l', lty=3, ylab=expression(paste(h[1](x))))
plot(x, Bsplines.basis[,2], type='l',lty=3, ylab=expression(paste(h[2](x))))
plot(x, Bsplines.basis[,3], type='l',lty=3, ylab=expression(paste(h[3](x)) ))
plot(x, Bsplines.basis[,4], type='l',lty=3, ylab=expression(paste(h[4](x))))
plot(x, Bsplines.basis[,5], type='l',lty=3, ylab=expression(paste(h[5](x))))
plot(x, Bsplines.basis[,6], type='l',lty=3, ylab=expression(paste(h[6](x))))
```



4.2.3 Natural Cubic Splines (NCS)

A cubic spline on $[a, b]$ is a **Natural Cubic Spline** if its *second* and *third* derivatives are *zero at a and b*. This condition implies that NCS is a linear function in the two extreme intervals $[a, \xi_1]$ and $[\xi_m, b]$. The linear functions in the two extreme intervals are completely determined by their neighboring intervals. The degrees of freedom of NCS's with m knots are:

$$4(m+1) - 3m - 4 = m$$

since we have 4 additional constraints (the ones on the boundary points).

Remark: For a curve estimation problem with data $(x_i, y_i)_{i=1}^n$, if we put n knots at the n data points (assumed to be unique), then using NCS we obtain a *smooth* curve passing through **all** y's.

Natural Cubic Spline

A Natural Cubic Spline with m knots is represented by m basis functions, for example one such basis is given by

$$\begin{aligned} N_1(x) &= 1 \\ N_2(x) &= x \\ N_{k+2}(x) &= d_k(x) - d_{k-1}(x) \end{aligned}$$

where

$$d_k(x) = \frac{(x-\xi_k)_+^3 - (x-\xi_m)_+^3}{\xi_m - \xi_k}$$

Each of these derivatives can be seen to have zero second and third derivative for $x \geq \xi_m$.

In `R` we can find the NCS as part of the `splines` library, by calling the `ns()` function. Specifically, we have that

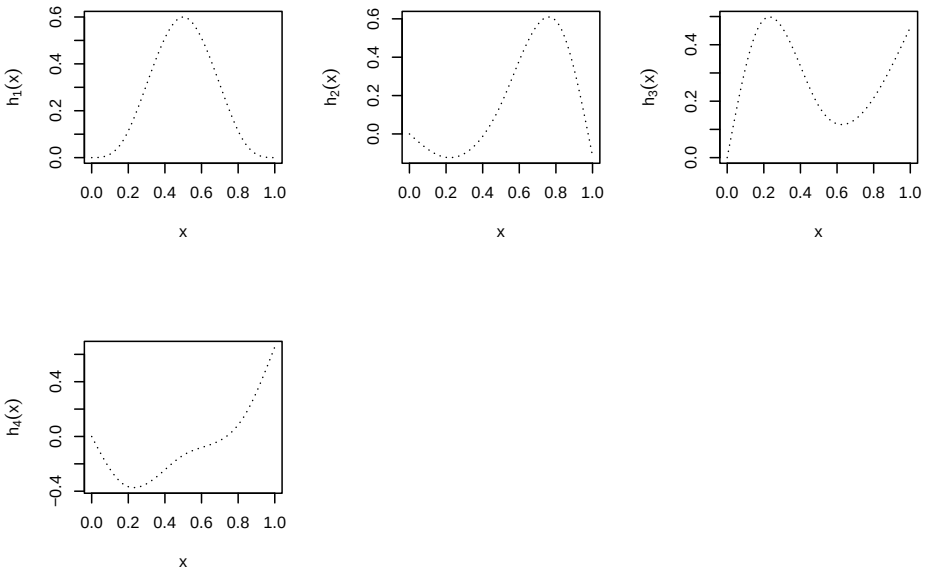
$$ns(x, df, knots, intercept = TRUE, Boundary.knots)$$

Natural Cubic Splines in `R`

```
NCS.basis =ns(x, knots=new.knots, Boundary.knots=c(0,1))
dim(NCS.basis)
```

```
## [1] 101   4
```

```
par(mfrow = c(2,3))
plot(x, NCS.basis[,1], type='l', lty=3, ylab=expression(paste(h[1](x))))
plot(x, NCS.basis[,2], type='l',lty=3, ylab=expression(paste(h[2](x))))
plot(x, NCS.basis[,3], type='l',lty=3, ylab=expression(paste(h[3](x)) ))
plot(x, NCS.basis[,4], type='l',lty=3, ylab=expression(paste(h[4](x))))
```



Recall that the linear functions in the two extreme intervals are completely determined by the other cubic splines. This means that data points in the two extreme intervals (i.e., outside the two boundary knots) are wasted since they do not affect the fitting. Therefore, by default, `R` puts the two boundary knots as the *min* and *max* of the x_i's.

As in the case of the `bs()` function:

- You can tell `R` the location of knots, which are the interior knots. Recall that a NCS with m knots has m df. So, the df is equal to the number of (interior) knots plus 2, where 2 means the two boundary knots.
- Or you can tell `R` the df. If intercept = TRUE, then we need $m = df - 2$ knots, otherwise we need $m = df - 1$ knots. Again, by default, `R` puts knots at the $1/(m+1), ..., m/(m+1)$ quantiles of $x_{1:n}$.

4.2.4 Regression Splines

Recall that for a given set of knots, the corresponding cubic splines form a *linear space* of functions with dimension $(m + 4)$. So, the **Regression Splines** use a basis expansion approach:

$$g(x) = \beta_1 h_1(x) + \beta_2 h_2(x) + ... + \beta_p h_p(x)$$

* If Cubic Splines are used as basis functions $p = m + 4$.

- If Natural Cubic Splines (NCS) are used as basis functions $p = m$.

We can represent the model on the observed n data points using matrix notation:

$$\begin{pmatrix} y_1 \\ y_2 \\ ... \\ y_n \end{pmatrix}_{n\times 1} = \begin{pmatrix} h_1(x_1) & ... & h_p(x_1) \\ h_1(x_2) & ... & h_p(x_2) \\ & & \\ h_1(x_n) & ... & h_p(x_n) \end{pmatrix}_{n\times p} \begin{pmatrix} \beta_1 \\ \beta_2 \\ ... \\ \beta_p \end{pmatrix}_{p\times 1}$$

where our *"design"* matrix is the matrix $\mathbf{F}$ of basis functions. Therefore, we can estimate the coefficients by solving the following Least-Squares problem:

$$\hat{} = \arg\min ||\mathbf{y} - \mathbf{F}\ ||^2$$

4.2.5 K-Fold Cross-Validation

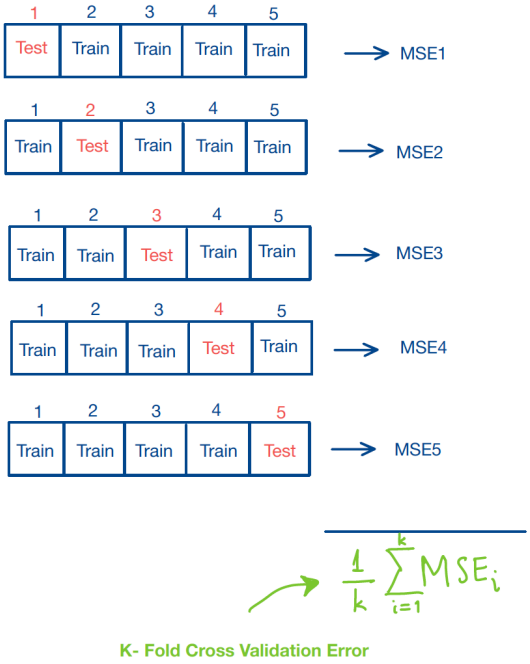
One of the challenges in using splines regression is the choice of the degrees of freedom/ knots. One way to optimally select the number of knots or degrees of freedom is the so-called K-fold cross-validation approach. This is outlined in the steps below:

1. Set a fixed number of knots (or df).
2. Divide the set of observations into k groups (or *folds*).
3. Leave the first fold as a validation set (not used to fit the model). Fit the Regression Spline with a fixed number of knots using the remaining $k-1$ folds.
4. Calculate the Mean Square Error for fold 1: MSE_1.
5. Repeat the previous steps k times. Each time a new validation set is used to calculate MSE_i.
6. Calculate the average k-fold Cross-Validation error:

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

7. Repeat 2 to 6 with a new number of knots (or df).
8. Select the number of knots that minimizes the k-fold CV error or $CV(k)$.

Then, we repeat the same process for a different number of knots and in the end we select the one that minimizes the $CV(k)$. A sketch of this approach is shown below:



4.3 Smoothing Splines

In Regression Splines, we need to choose the number and the location of knots (or the degrees of freedom). As we discussed, B-splines and NCS are both methods that construct a $n \times p$ basis matrix $\mathbf{F}$, and then model the outcome using a linear regression on $\mathbf{F}$. Inevitably, we need to select the order of the spline, the number of knots (AIC, BIC, CV) and even the location of knots, which is a quite challenging task. So, we need to consider whether there is an alternative approach that we can use to select the number and location of knots automatically.

A Smoothing Spline is a spline designed to balance fit with smoothness, and it starts by suggesting a very bad solution to our problem: by putting knots at *all* the observed data points $(x_1, \dots, x_n)$:

$$\mathbf{y}_{n\times 1} = \mathbf{F}_{n\times p}\beta_{p\times 1}$$

Then, we can construct n NCS basis. However, we know that with this approach we are going to run into overfitting problems. So, instead of selecting the knots, we recall last week's discussion on penalizing models with many parameters. This leads to minimizing the following objective functions with a ridge-type shrinkage:

$$\min_{\beta}\Big\{||\mathbf{y} - \mathbf{F}\beta||^2 + \lambda\beta^T\Omega\beta\Big\}$$

where the tuning parameter λ is often chosen by CV (Ω will be defined later).

So, now we want to understand whether solving such a minimzation problem can provide us with a good solution with desirable properties.

4.3.1 The Roughness Penalty Approach

Let $S[a, b]$ be the space of all "smooth'' functions defined on $[a, b]$. This is a second order Sobolev space where global polynomial functions and cubic splines functions live ($S[a, b]$ is an infinite-dimensional function space. Our goal is to find the best function in $S[a, b]$ to approximate f.

Let us consider solving the following Penalized Residual Sum of Squares problem:

$$RSS_\lambda(g) = \sum_{i=1}^{n}[y_i - g(x_i)]^2 + \lambda \int_a^b [g''(x)]^2 dx.$$

where λ is a smoothing parameter. The first term measures the closeness of the model to the data, while the second term penalizes the roughness/curvature of the function. We solve this problem on $[a, b] = [\min x_i, \max_i]$ for functions with finite roughness penalty $\int_a^b [g''(x)]^2 dx < \infty$. This is known as a second order Sobolev space.

Note that $\int_a^b [g''(x)]^2 dx$ is called the **roughness penalty**.

From the expression above, we can see that λ is the smoothing parameter that controls the bias-variance trade-off. Therefore, when $\lambda = 0$, we interpolate the data and that leads us to overfitting. When $\lambda = \infty$, then we return to linear least-squares regression. It turns out that the solution to the penalized residual sum of squares has to be a NCS. Indeed,

Theorem

$$\min_g RSS_\lambda(g) = \min_{\tilde{g}} RSS_\lambda(\tilde{g})$$

} where $\tilde{g}$ is a NCS with knots at the n data points.

Let's call $g(x)$ the as the optimal solution. Since the loss part in $RSS_\lambda(g)$ only involves n data points, we can find define a natural cubic spline (NCS) fit that we can call $\tilde{g}(x)$ such that it matches $g(x)$ at the observations x_i, $i = 1, \dots, n$, i.e.

$$g(x_i) = \tilde{g}(x_i), i = 1, \dots, n$$

We can always find such $\tilde{g}$ since our space consists of n basis. Then, we can show that

$$\int g''^2 dx \geq \int \tilde{g}''^2 dx$$

meaning that we will *always* prefer the $\tilde{g}$, the NCS "representation'' of g, since the *penalty* is smaller, and the *loss* doesn't change.

4.3.2 Proof

To establish the Theorem, we essentially need to show that

$$\int g''^2 dx \geq \int \tilde{g}''^2 dx$$

Thus, we define $h(x) = g(x) - \tilde{g}(x)$ for which we know that $h(x_i) = 0$ for $i = 1, \ldots, n$. Then

$$\int g''^2\, dx = \int \tilde{g}''^2\, dx + \int h''^2\, dx + 2\int \tilde{g}'' h''\, dx$$

and without loss of generality assuming that the x_is are ordered, we obtain:

$$\begin{aligned}
\int \tilde{g}'' h''\, dx &= \tilde{g}'' h' \Big|_a^b - \int_a^b h' \tilde{g}^{(3)}\, dx \\
&= -\sum_{i=1}^{n-1} \tilde{g}^{(3)}\left(x_j^+\right) \int_{x_j}^{x_{j+1}} h'\, dx \quad (\tilde{g}^{(3)} \text{ constant piecewise}) \\
&= -\sum_{i=1}^{n-1} \tilde{g}^{(3)}\left(x_j^+\right) \left(h(x_{j+1}) - h(x_j)\right)
\end{aligned}$$

The second equation is because $\tilde{g}$ is a NCS and therefore we know that it has zero second derivative on the two boundaries a and b. The third equation is true, because $\tilde{g}$ is at most a 3rd order polynomial on any region and as a consequence has constant third derivatives, which we can pull out of the integration. The last equation holds because we said that $h(x) = 0$ on all the observation points x_i. Therefore, this shows that the roughness penalty of our NCS solution is no larger than the best solution g. If we also take into account that $\tilde{g}$ is also in the space $S[a, b]$, then g must be our NCS solution.

4.4 Fitting Smoothing Splines

From the previous discussion, we conclude that g has a **finite sample representation**

$$\hat{g}(x) = \sum_i \beta_i N_i(x)$$

where N_i are a set of NCS basis functions with knots at each of the x *unique* values. Therefore, the penalty function becomes

$$\begin{aligned}
\int_a^b g''^2 dx &= \int \left(\sum_i \beta_i N_i''(x)\right)^2 dx \\
&= \sum_{i,j} \beta_i \beta_j \int N_i''(x) N_j''(x) dx \\
&= \beta^T \Omega \beta
\end{aligned}$$

where $\Omega_{n\times n}$ with $\Omega_{ij} = \int N_i^{''}(x)N_j^{''}(x)dx$.

Hence our goal is to find β that minimizes

$$RSS_\lambda(\beta) = (\mathbf{y} - \mathbf{F}\beta)^T(\mathbf{y} - \mathbf{F}\beta) + \lambda\beta^T\Omega\beta$$

This is a ridge penalized function and the solution is

$$\begin{aligned}\hat{\beta} &= \arg\min_\beta RSS_\lambda(\beta) \\ &= (\mathbf{F}^T\mathbf{F} + \lambda\Omega)^{-1}\mathbf{F}^T\mathbf{y}\end{aligned}$$

The smoothing spline version of the "hat'' matrix is called the **smoother matrix**

$$\hat{\mathbf{y}} = \mathbf{F}(\mathbf{F}^T\mathbf{F} + \lambda\Omega)^{-1}\mathbf{F}^T\mathbf{y} = S_\lambda\mathbf{y}$$

Remarks

We have done the analysis of degrees of freedom for ridge type regression. The degrees of freedom of a smoothing spline are

$$df = tr(S_\lambda)$$

which ranges between 0 and n. Under some special constructions (Demmler and Reinsch, 1975), a basis with double orthogonality property, can lead to

$$\mathbf{F}^T\mathbf{F} = \mathbf{I}, \Omega = diag(d_i)$$

where d_is are arranges in an increasing order and in addition $d_2 = d_1 = 0$. Using this basis we have

$$\hat{\beta} = (\mathbf{F}^T\mathbf{F} + \lambda\Omega)^{-1}\mathbf{F}^T\mathbf{y} = \Big(\mathbf{I} + \lambda diag(d_i)\Big)^{-1}\mathbf{F}^T\mathbf{y}$$

i.e.

$$\hat{\beta}_i = \frac{1}{1+\lambda d_i}\hat{\beta}_i^{LS}$$

Choosing λ

Choosing the penalty λ is the same as in ridge regression.

– Leave-one-out CV:

$$\mathrm{CV}(\lambda) = \frac{1}{n}\sum_{i=1}^n\left(\frac{y_i - \hat{g}(x_i)}{1 - S_\lambda(i,i)}\right)^2.$$

– Generalized CV:

$$\mathrm{GCV}(\lambda) = \frac{1}{n}\sum_{i=1}^n\left(\frac{y_i - \hat{g}(x_i)}{1 - \frac{1}{n}\mathrm{tr}(S_\lambda)}\right)^2.$$

4.5 The Birthrates Example in R

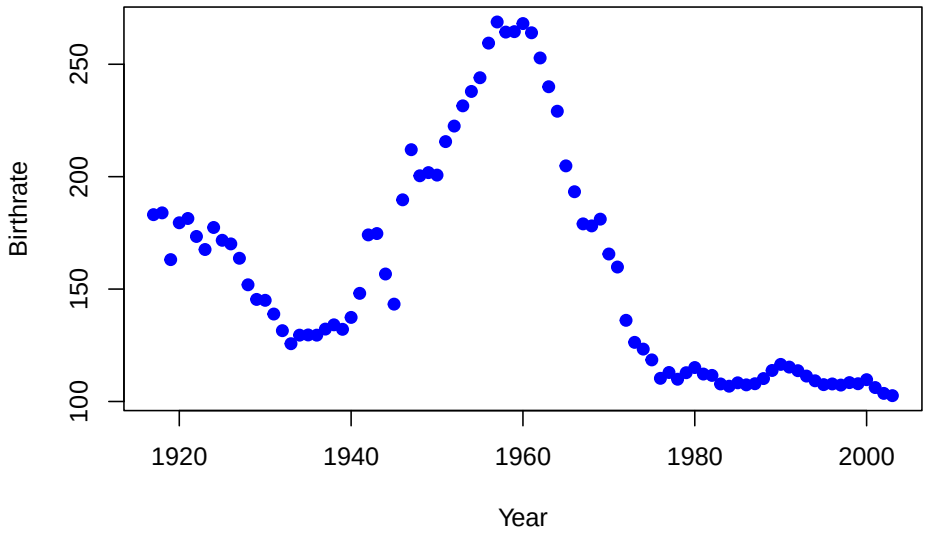
The Birthrate data set contains U.S. birth rate data from 1917 to 2003.

```
birthrates= read.csv("data/week4/birthrates.csv")
head(birthrates)
```

```
##   Year Birthrate
## 1 1917     183.1
## 2 1918     183.9
## 3 1919     163.1
## 4 1920     179.5
## 5 1921     181.4
## 6 1922     173.4
```

Plotting the data we have

```
plot(birthrates, pch = 19, col = "blue")
```

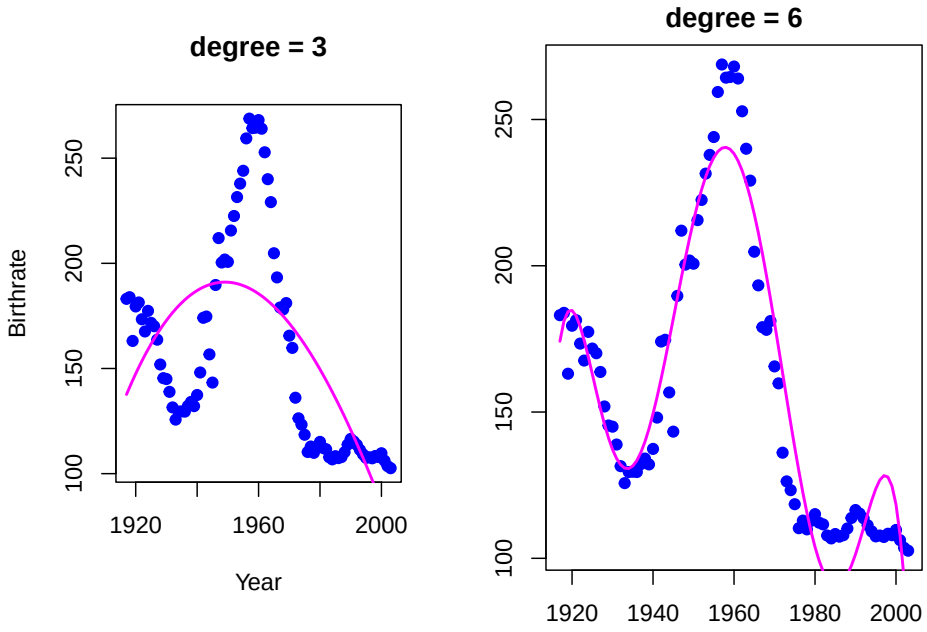


which is an highly nonlinear trend.

We first try to fit a polynomial regression with orthogonal polynomials for polynomials with degrees 3, and 6. The fitted lines can be shown below:

```
par(mfrow=c(1,2))

    poly3.fit <- lm(Birthrate ~ poly(Year, 3), data = birthrates)
    plot(birthrates, pch = 19, col = "blue")
    lines(birthrates$Year, poly3.fit$fitted.values, lty = 1, col = "magenta", lwd = 2)
    title("degree = 3")

    par(mar = c(2,3,2,0))
```

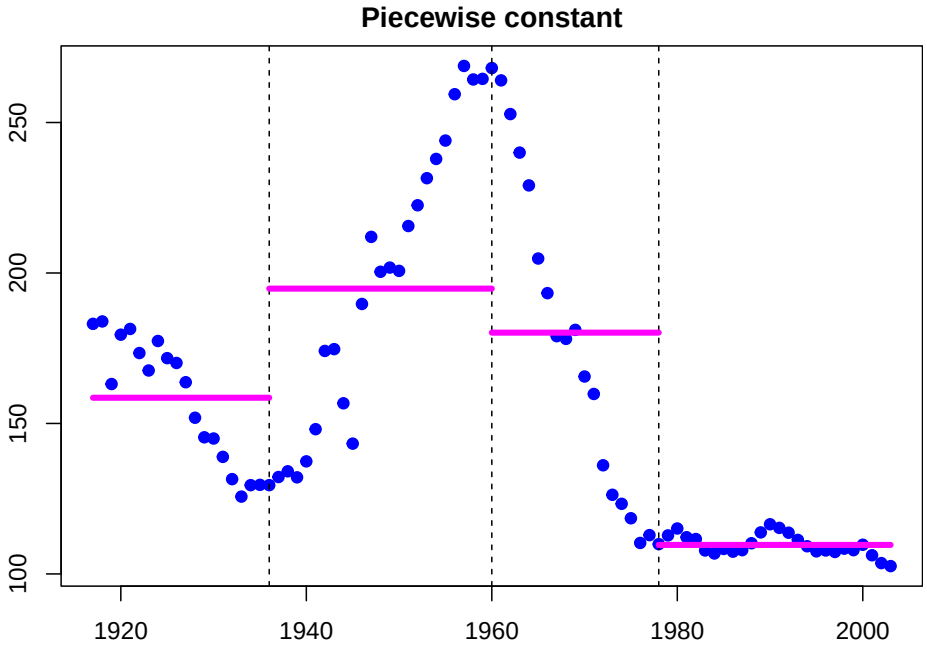
```
    poly6.fit <- lm(Birthrate ~ poly(Year, 6), data = birthrates)
    plot(birthrates, pch = 19, col = "blue")
    lines(birthrates$Year, poly6.fit$fitted.values, lty = 1, col = "magenta", lwd = 2)
    title("degree = 6")
```



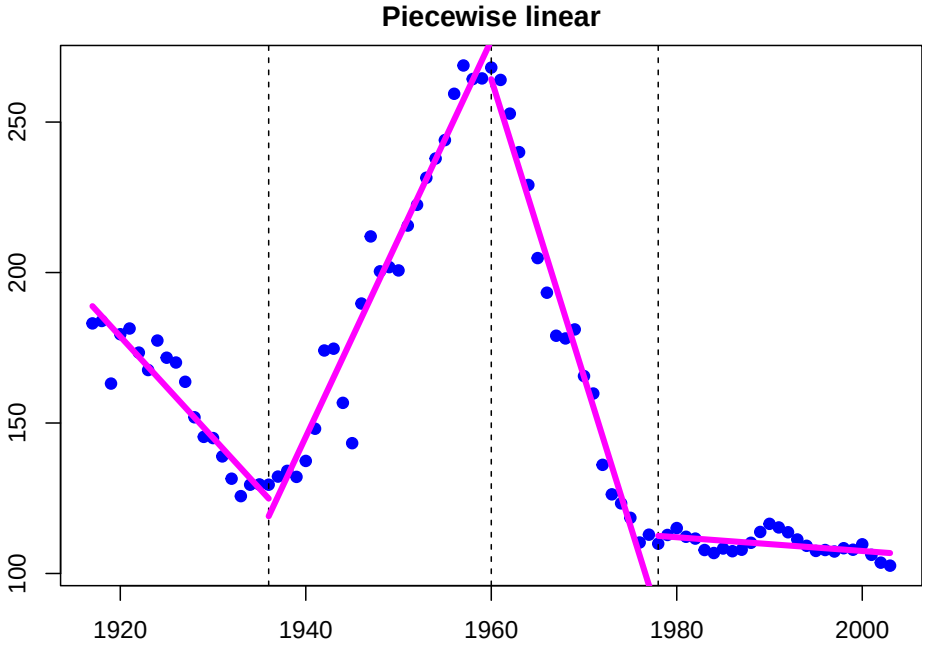
We can see that the 6th order polynomial is better, but it is not very good. We can try fitting local polynomials - piecewise polynomials of lower order (we try constant functions and linear function in the example below:

```
myknots = c(1936, 1960, 1978)

bounds = c(1917, myknots, 2003)

# Piecewise Constant Polynomials
   mybasis = cbind("x_1" = (birthrates$Year < myknots[1]),
                   "x_2" = (birthrates$Year >= myknots[1])*(birthrates$Year < myknots
                   "x_3" = (birthrates$Year >= myknots[2])*(birthrates$Year < myknots
                   "x_4" = (birthrates$Year >= myknots[3]))

    const.fit <- lm(birthrates$Birthrate ~ . -1, data = data.frame(mybasis))
    par(mar = c(2,3,2,0))
    plot(birthrates, pch = 19, col = "blue")
    abline(v = myknots, lty = 2)
    title("Piecewise constant")

    for (k in 1:4)
```

```
    points(c(bounds[k], bounds[k+1]), rep(const.fit$coefficients[k], 2), type = "l", lty = 1,
```




```
# Piecewise Linear Polynomials
mybasis = cbind("x_1" = (birthrates$Year < myknots[1]),
                "x_2" = (birthrates$Year >= myknots[1])*(birthrates$Year < myknots[2]),
                "x_3" = (birthrates$Year >= myknots[2])*(birthrates$Year < myknots[3]),
                "x_4" = (birthrates$Year >= myknots[3]),
                "x_11" = birthrates$Year*(birthrates$Year < myknots[1]),
                "x_21" = birthrates$Year*(birthrates$Year >= myknots[1])*(birthrates$Year < m
                "x_31" = birthrates$Year*(birthrates$Year >= myknots[2])*(birthrates$Year < m
                "x_41" = birthrates$Year*(birthrates$Year >= myknots[3]))

line.fit <- lm(birthrates$Birthrate ~ .-1, data = data.frame(mybasis))
par(mar = c(2,3,2,0))
plot(birthrates, pch = 19, col = "blue")
abline(v = myknots, lty = 2)
title("Piecewise linear")

for (k in 1:4)
    points(c(bounds[k], bounds[k+1]), line.fit$coefficients[k] + c(bounds[k], bounds[k+1])*li
           type = "l", lty = 1, col = "magenta", lwd = 4)
```

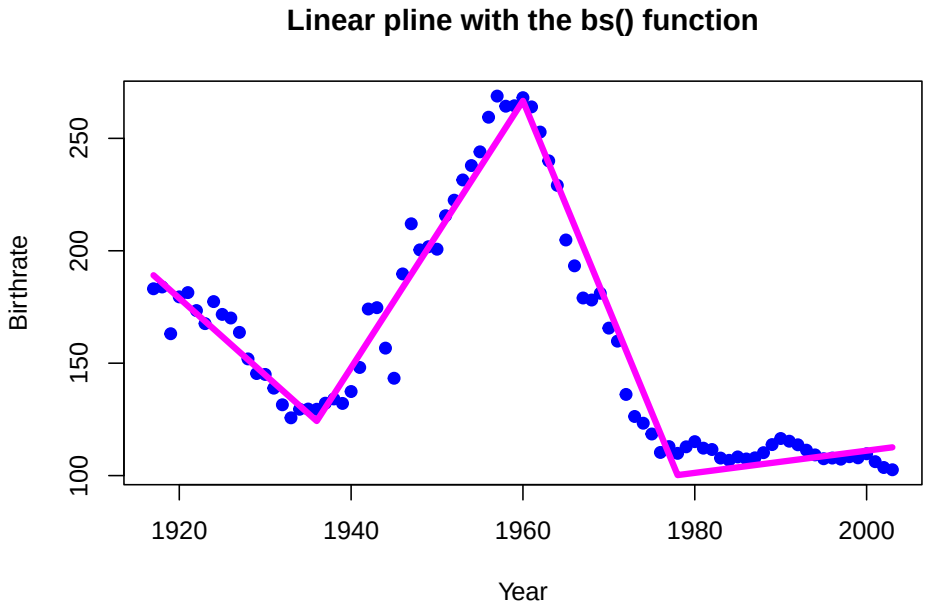


As we discussed in the lecture, it is a localized fit, but there are discontinuities and the overall fit is not good.

4.5.0.1 Splines Regression

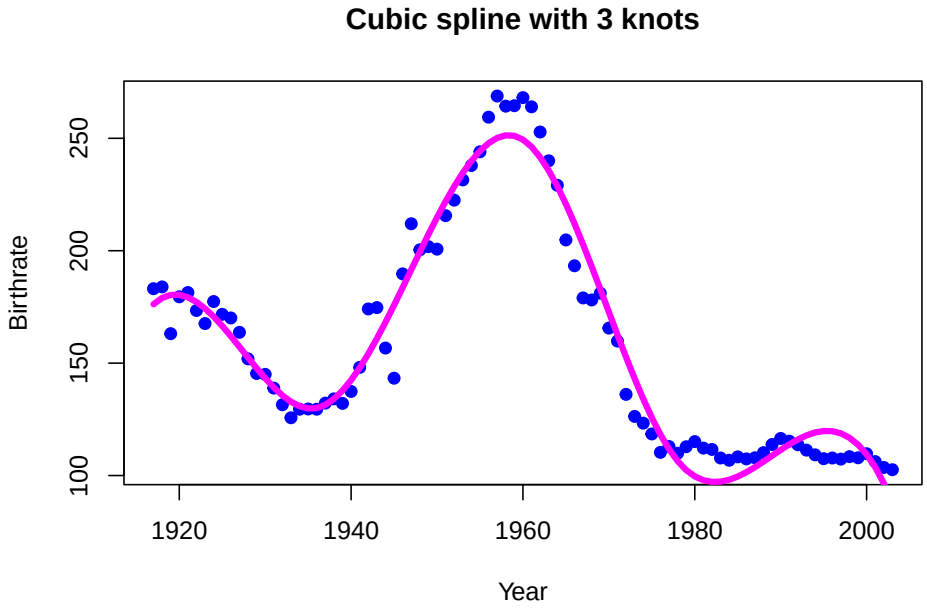
Instead, we can use **splines** to fit the data. Let us first try to use the B-Splines. We will first try a **linear spline**

```
bsplines.lin.fit <- lm(Birthrate ~ splines::bs(Year, degree = 1, knots = myknots), data
    plot(birthrates, pch = 19, col = "blue")
    lines(birthrates$Year, bsplines.lin.fit$fitted.values, lty = 1, col = "magenta", lw
    title("Linear pline with the bs() function")
```



and then a **cubic spline** (as the ones discussed in class)

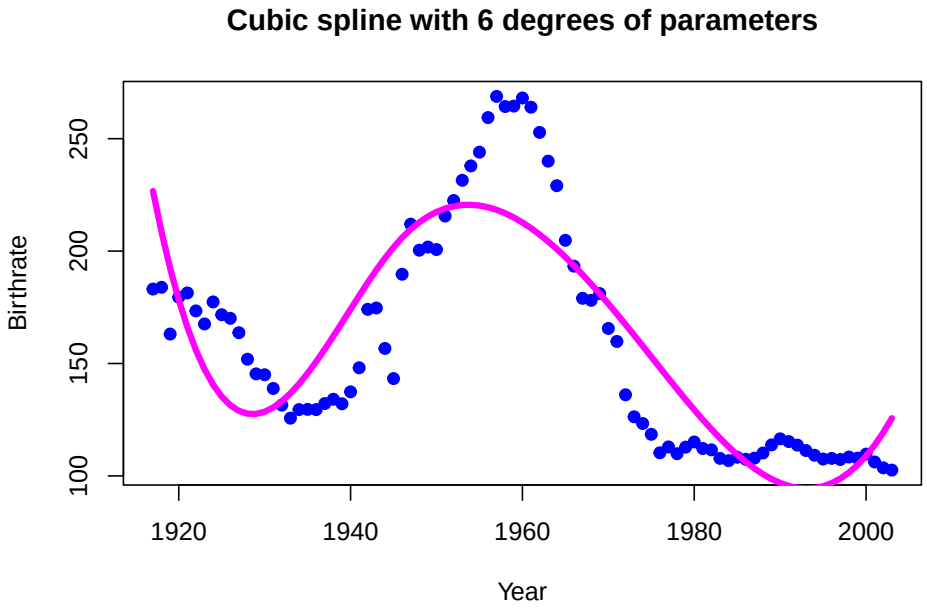
```
bsplines.3.fit <- lm(Birthrate ~ splines::bs(Year, degree = 3, knots = myknots), data = birthrate
    plot(birthrates, pch = 19, col = "blue")
    lines(birthrates$Year, bsplines.3.fit$fitted.values, lty = 1, col = "magenta", lwd = 4)
    title("Cubic spline with 3 knots")
```



In the plot below, we used our pre-defined knots. We can alternatively determine

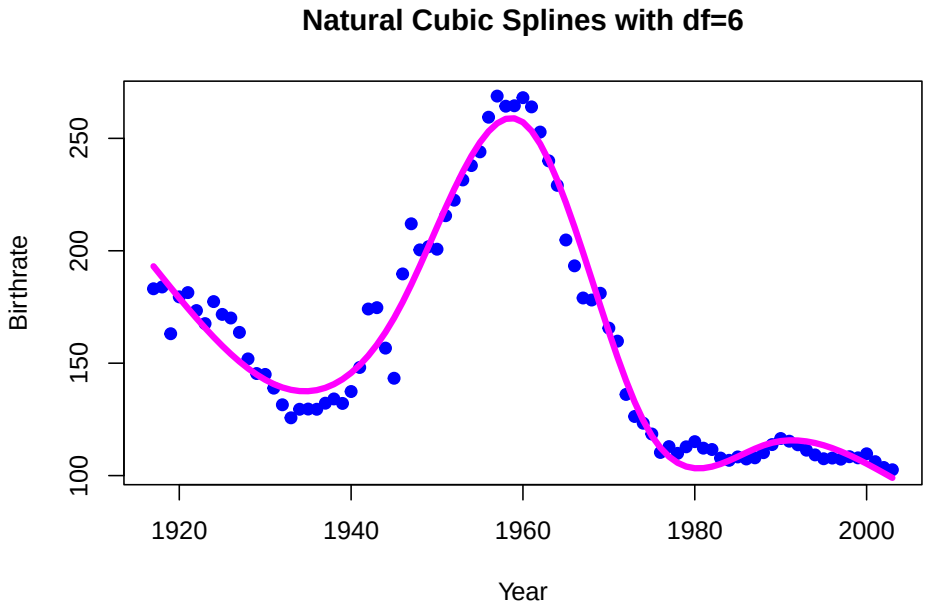
the degrees of freedom in the function `bs`:

```
bsplines.3.fit.new <- lm(Birthrate ~ splines::bs(Year, df = 5), data = birthrates)
    plot(birthrates, pch = 19, col = "blue")
    lines(birthrates$Year, bsplines.3.fit.new$fitted.values, lty = 1, col = "magenta",
    title("Cubic spline with 6 degrees of parameters")
```



Let us now try the **Natural Cubic Splines**

```
library(splines)
    ns.splines.fit = lm(Birthrate ~ ns(Year, df=6), data=birthrates)
    plot(birthrates, pch = 19, col = "blue")
    lines(birthrates$Year, ns.splines.fit$fitted.values, lty = 1, col = "magenta", lwd
    title("Natural Cubic Splines with df=6")
```



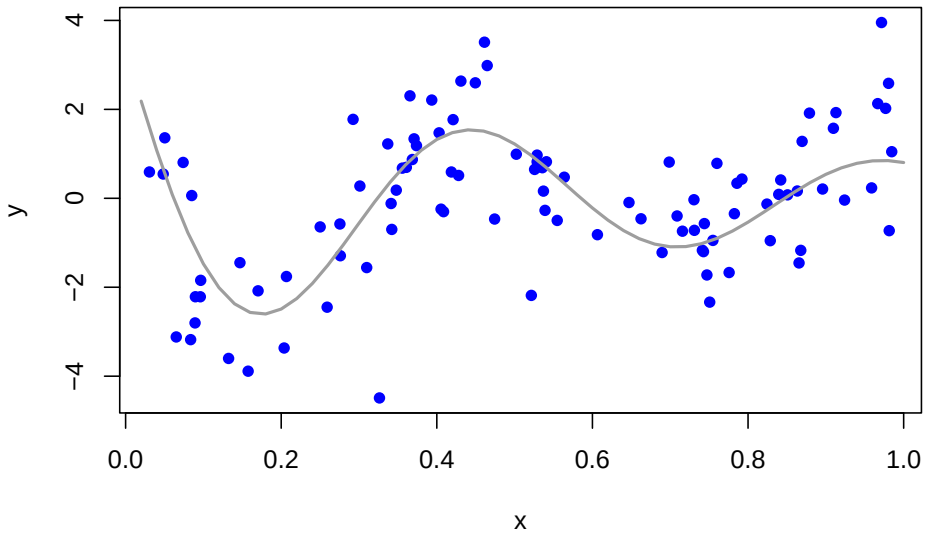
4.5.0.2 A Simulated Example of Smoothing Splines

In this example, we are going to reproduce the example from the ESL book in section 5.5.2 in which we the true function of the data is given by

$$f(x) = \frac{sin(12(x+0.2))}{x+0.2}, \, x \in [0,1]$$

So, we are first going to simulate data from this curve:

```
set.seed(598)
n=100

x = sort(runif(n))
y = sin(12*(x+0.2))/(x+0.2) + rnorm(n, 0, 1)

plot(x, y, col="blue", pch=16 )

funf_x = 1:50/50
funf_y = sin(12*(funf_x+0.2))/(funf_x+0.2)
lines(funf_x, funf_y, col=8, lwd=2)
```



4.5.0.3 Fitting a Smoothing Spline model to the simulated data above

The function to use is `smooth.spline` and is part of the `splines` library in `R`.

```
library(splines)
?smooth.spline
```

The model is fitted as follows:

```
spline.model = smooth.spline(x, y, df=5)
spline.model
```

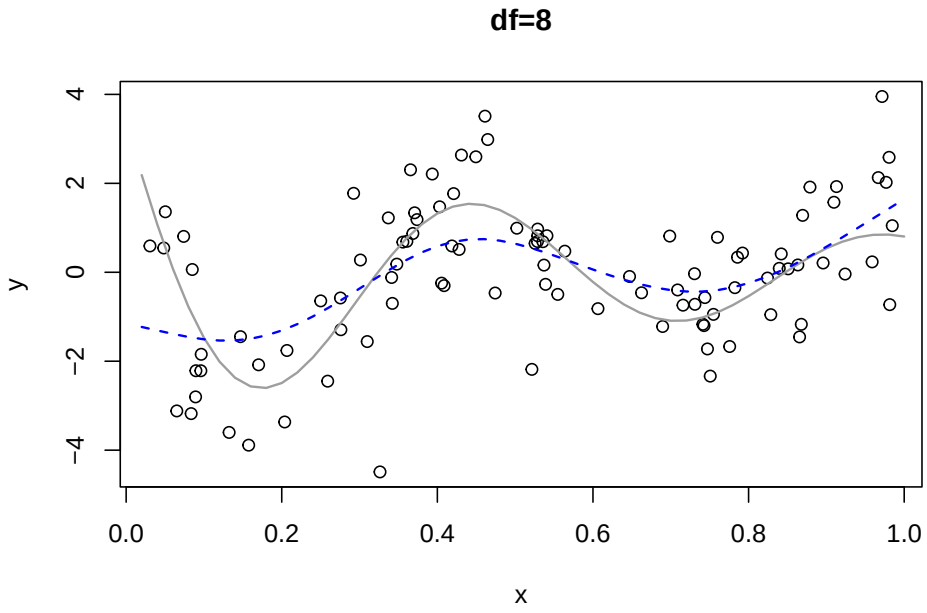
```
## Call:
## smooth.spline(x = x, y = y, df = 5)
##
## Smoothing Parameter  spar= 0.9616029  lambda= 0.006048723 (12 iterations)
## Equivalent Degrees of Freedom (Df): 5.000634
## Penalized Criterion (RSS): 162.0406
## GCV: 1.795488
```

We can compute the fitted values as follows:

```
fitted.y = predict(spline.model, funf_x)
```

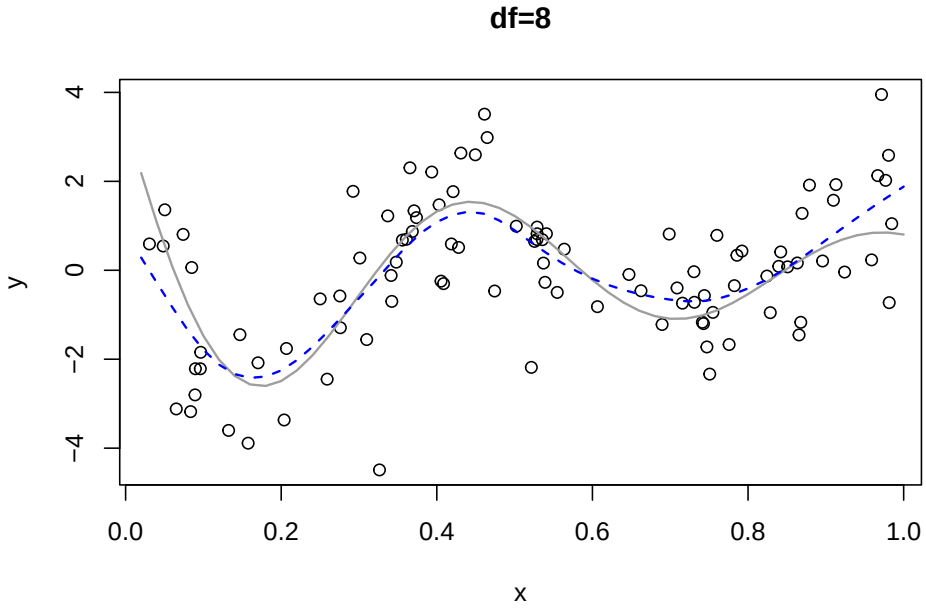
To illustrate the fit of the model, we can plot the fitted line on top of the data:

```
plot(x,y, xlab='x', ylab='y');
lines(funf_x, funf_y, col=8, lwd=1.5);
lines(fitted.y,  lty=2, col='blue', lwd=1.5);
title('df=8');
```



The blue line is the true model, while the gray line is the fitted model with 5 degrees of freedom. As we discussed, if we change the degrees of freedom, e.g. df=8, then the line will be more/less sensitive to the data. For example for df=8 we have

```
plot(x,y, xlab='x', ylab='y');
lines(funf_x, funf_y, col=8, lwd=1.5);
lines(predict(smooth.spline(x, y, df=8), funf_x),  lty=2, col='blue', lwd=1.5);
title('df=8');
```



4.5.0.4 Choice of Lambda

Many R packages come with an inbuilt option for determining lambda, primarily based on leave-one-out cross-validation (LOOCV) and generalized cross-validation (GCV).

There's no need to specify lambda directly. Instead, we indicate the desired degrees of freedom (df), ranging from 0 to n. `R` then determines the appropriate lambda. We can then consult both CV and GCV curves to decide the best lambda or df.

```
model.fit = smooth.spline(x, y, df=9);
```

When using the `smooth.spline` function with a degree of freedom set at 9, the returned value `model.fit$df` may have a slight deviation due to rounding errors.

The leverage output is equivalent to the diagonal entries of the smoothing matrix.

```
model.fit$df
```

```
## [1] 9.001417
```

```
model.fit$lev  # leveage = diagonal entries of the smoother matrix
```

```
##   [1] 0.30893986 0.16530188 0.15452511 0.10279673 0.08811473 0.08173143
##   [7] 0.08148178 0.08160942 0.08169108 0.08413540 0.08441745 0.11256266
##  [13] 0.12120898 0.12602066 0.13175493 0.13750648 0.13745737 0.11875674
```

```
##  [19] 0.11154402 0.10015615 0.09976906 0.08986948 0.08508903 0.08020005
##  [25] 0.07203883 0.06701137 0.06528799 0.06497591 0.06309976 0.06147779
##  [31] 0.06087571 0.06065672 0.06070371 0.06080451 0.06105153 0.06369195
##  [37] 0.06523756 0.06573835 0.06643670 0.06911934 0.06984682 0.07231683
##  [43] 0.07347632 0.08032413 0.08330237 0.08402559 0.08427316 0.07806223
##  [49] 0.07013561 0.06947727 0.06946921 0.06948635 0.06949033 0.07114094
##  [55] 0.07177155 0.07287487 0.07400078 0.08729462 0.09996053 0.14342850
##  [61] 0.14982974 0.13605679 0.10479241 0.09474069 0.08413352 0.07762692
##  [67] 0.06748810 0.06720784 0.06346315 0.06318782 0.06301969 0.06275533
##  [73] 0.06281727 0.06333844 0.06451613 0.06999234 0.07288373 0.07423106
##  [79] 0.07646914 0.07952440 0.07869513 0.07674170 0.07634088 0.07528017
##  [85] 0.07601450 0.07640336 0.07689899 0.07734393 0.08037947 0.08796840
##  [91] 0.09303429 0.09391332 0.09611884 0.10022513 0.10988799 0.12007621
##  [97] 0.13650369 0.15233032 0.15631587 0.17386379
```

```
sum(model.fit$lev)
```

```
## [1] 9.001417
```

By default, the function offers a GCV score, but you can alter it to LOOCV for comparison.

```
model.fit$cv  # default: GCV
```

```
## [1] FALSE
```

```
sum((y-model.fit$y)^2)/(1-model.fit$df/n)^2/n
```

```
## [1] 1.577447
```

```
fit=smooth.spline(x, y, df=9, cv=T) # set 'cv=T' to return CV
fit$cv
```

```
## [1] TRUE
```

```
sum(((y-fit$y)/(1-fit$lev))^2)/n
```

```
## [1] 1.58123
```