6 PQHS 471 Notes Week 6

6.1 Week 6 Day 1

6.1.1 ISLR 6.3 (Cont'd)

- 6.3.1: Principal components regression (PCR): Use the first M PCs of the input variables as the predictors.

$$y = \beta_0 + \beta_1 \text{PC}_1 + \cdots + \beta_M \text{PC}_M.$$

  - The hyperparameter M can be chosen with cross-validation, or with a pre-determined threshold for the fraction of total variance explained by the first M PCs.
  - The PCs are determined solely on X. The outcome variable is not involved in PC calculation.
  - PCR is not feature selection. It is more like feature extraction.
  - There is no guarantee that the directions that best explain the predictors will also be the best directions to use for predicting the response.
  - It performs well when the first few PCs capture most of the variance in the predictors.
  - It can be viewed as a discrete version of ridge regression.

The R `pls` package provides a function `pcr()` for PC regression. The output is of class `mvr`. So, to look for the documents for `summary()` and `predict()`, look at `summary.mvr()` and `predict.mvr()`.

```
require(pls)
library(ISLR)
Hitters = na.omit(Hitters)

pcr.fit = pcr(Salary ~ ., data=Hitters, scale=T)
coef(pcr.fit, ncomp=5)  ## coefficients for the standardized PREDICTORs for the model using 5 PCs
pcr.pred = predict(pcr.fit, Hitters, ncomp=5)  ## prediction using 5 PCs

## Using all PCs leads to the same model as the full data linear model
mod = lm(Salary ~ ., data=Hitters)  ## full data linear model
mod.coef = coef(mod)[-1]
mod.pred = predict(mod, Hitters)

pcr.coef = coef(pcr.fit, ncomp=19) / apply(data.matrix(Hitters[,-19]), 2, sd)
pcr.pred = predict(pcr.fit, Hitters, ncomp=19)
all.equal(mod.coef, as.numeric(pcr.coef), check.attributes = F)  ## True
all.equal(mod.pred, as.numeric(pcr.pred), check.attributes = F)  ## True

pcr.fit = pcr(Salary ~ ., data=Hitters, scale=T, validation="CV")
summary(pcr.fit)  ## summary.mvr()
validationplot(pcr.fit, val.type="RMSE")
```

- 6.3.2: Partial least squares (PLS):

1. Standardize all the features. Then let $y_0 = y$, $x_{0,j} = x_j$ $(j = 1, \ldots, p)$.
2. For $m = 1, \ldots, M$:
   a. Perform simple linear regression of y_{m-1} on $x_{m-1,j}$ to obtain slope $\hat{\beta}_{mj}$ $(j = 1, \ldots, p)$.
   b. Compute $Z_m = \sum_j \hat{\beta}_{mj} x_j$.
   c. Perform simple linear regression of y_{m-1} on Z_m to obtain residual y_m, and of $x_{m-1,j}$ on Z_m to obtain residual $x_{m,j}$ $(j = 1, \ldots, p)$.
3. Perform the final regression:

$$y = \beta_0 + \beta_1 Z_1 + \cdots + \beta_M Z_M.$$

   - The hyperparameter M can be chosen with cross-validation.
   - Z_m can be viewed as a composite score over individual contributions from the predictors.

The R `pls` package provides a function `plsr()` for partial least squares regression.

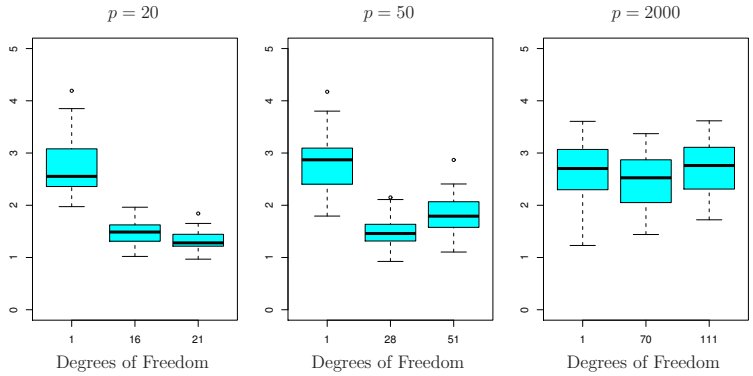
```
pls.fit = pcr(Salary ~ ., data=Hitters, scale=T)
coef(pls.fit, ncomp=5)  ## coefficients for the PREDICTORs for the model using 5 PCs
pcr.pred = predict(pls.fit, Hitters, ncomp=5)  ## prediction using 5 PCs

## Using all p Z-scores leads to the same model as the full data linear model
pls.coef = coef(pls.fit, ncomp=19) / apply(data.matrix(Hitters[,-19]), 2, sd)
pls.pred = predict(pls.fit, Hitters, ncomp=19)
all.equal(mod.coef, as.numeric(pls.coef), check.attributes = F)  ## True
all.equal(mod.pred, as.numeric(pls.pred), check.attributes = F)  ## True

pls.fit = plsr(Salary ~ ., data=Hitters, scale=T, validation="CV")
summary(pls.fit)
validationplot(pls.fit, val.type="MSEP")
```

6.1.2 ISLR 6.4 High dimensional data

- 6.4.1: High dimensional data have a large p. The authors are too strict by using $n < p$ as the definition.
- 6.4.2: Traditional measures such as C_p, AIC, and BIC are not suitable for high dimensional data.
- 6.4.3: Curse of dimensionality:



- 6.4.4: Do not over-interpret the results.

6.1.3 HOML 4

Linear regression with least squares:

- The "Normal Equation", $\hat{y} = (X'X)^{-1}X'y$, is the mathmatical expression of the least squares solution. A mathmatical formula implemented literally may not be a robust or fast algorithm. In statistics software, Cholesky decomposition of $X'X$ is used to avoid matrix inversion.
- But even with the Choleskey decomposition, a very large p can be a challenge.

Cost functions:

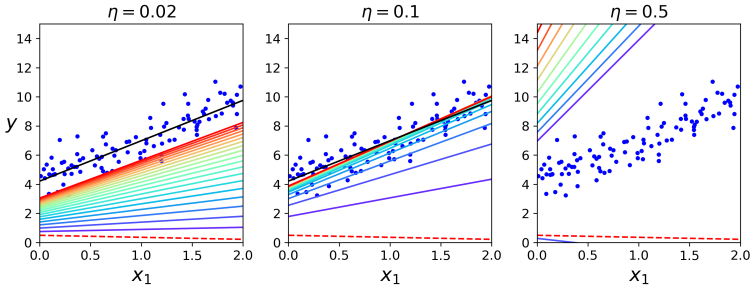
- $C(\theta) = \frac{1}{n}\sum_i C(y_i, f(x_i;\theta))$, where $f(x_i;\theta)$ is a prediction model as a function of predictors x_i and parameters θ. It is called a loss function in Statistics. Average cost is used because it keeps the scale when doing SGD.
- For continuous outcomes, a common choice is the **mean squared error**, $\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{n}\text{RSS}$. Minimizing MSE is equivalent to minimizing RSS or to minimizing RMSE $= \sqrt{\text{MSE}}$.
- In general, wherever we can define a likelihood function $L(y_i;\theta)$, we can define a cost function $c_i(\theta) = -\log L(y_i;\theta)$. Maximizing $\prod_i L(y_i;\theta)$ is equivalent to minimizing $C(\theta) = \sum_i c_i(\theta)$.
  - In logistic regression for binary outcomes, the likelihood for observation i is $p_i^{y_i}(1-p_i)^{1-y_i}$. The corresponding cost function is $c_i = -[y_i \log(p_i) + (1-y_i)\log(1-p_i)]$, which is called **cross entropy**.

  - Same is true for multinomial outcomes with k categories. The likelihood for $y_i = (y_{i1}, \ldots, y_{ik})$ ($y_{ij} = 1$ for a j and 0 for all other categories) is $\prod_j p_{ij}^{y_{ij}}$. The corresponding cross entropy is $c_i = -\sum_j y_{ij} \log(p_{ij})$.

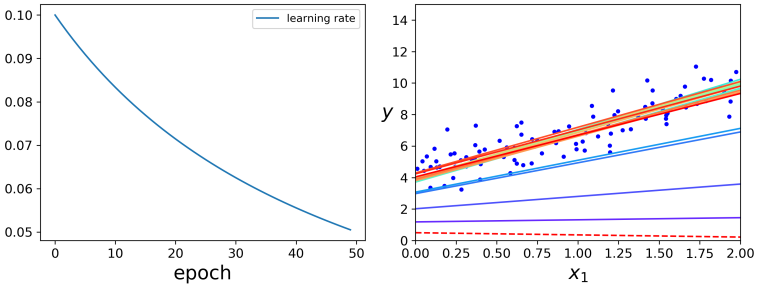
Evaluation criterion (performance measure) vs. **optimization criterion** (cost function): For a problem, we often have a desired evaluation criterion (a performance measure). Occasionally, the performance measure may not be easy to optimize directly. In that case, we may use a different optimization criterion (a cost function) to fit the model and use the evaluation criterion to evaluate the fitted model. For example, misclassification rate is often the evaluation criterion in classification but it is often not used as the target during optimization.

Gradient descent is a general iterative approach to minimizing a function $C(\theta)$.

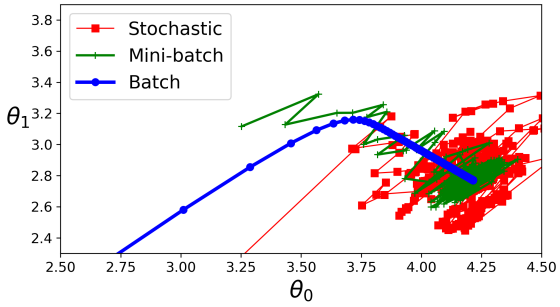
- The **gradient** of a function $C(\theta_1, \ldots, \theta_p)$ is a vector of partial derivatives $\nabla C = (\frac{\partial C}{\partial \theta_1}, \ldots, \frac{\partial C}{\partial \theta_p})^T$.
  - In linear regression, the gradient vector of MSE is $\nabla C = \frac{2}{n} X'(X\theta - y)$.
- **The algorithm**: (1) Initialize θ. (2) In every iteration, calculate ∇C with the current θ values, and update $\theta \leftarrow \theta - \eta \nabla C$, where $\eta > 0$ is the *learning rate* (which should be small).
- **Rationale**: To minimize $C(\theta)$ iteratively, we seek to identify a small change in θ, $\Delta\theta$, so that the reduction from $C(\theta)$ to $C(\theta + \Delta\theta)$ is the greatest. Since $C(\theta + \Delta\theta) - C(\theta) \approx (\nabla C)^T \Delta\theta$, the direction of fastest change is $-\nabla C$; that is, to *descend along the gradient*.
- **Batch gradient descent**: When n is not large, we calculate ∇C using all observations.
  - *Hyperparameters*: learning rate, number of iterations.
- **Stochastic gradient descent** (SGD): When n is very large, the calculation of ∇C can be slow. We can estimate it using a subset of the data. A **mini-batch size** is chosen (say, 100). The training set is randomly partitioned into subsets of the mini-batch size. We iterate through all the subsets. After going through the whole dataset, we have finished one **epoch**. Then we repartition the training set and repeat the process.
  - *Hyperparameters*: number of epochs, mini-batch size, learning rate (or parameters in a learning schedule)
  - Note that the author calls the SGD with mini-batch size 1 a "stochastic gradient descent", and the SGD with mini-batch size >1 a "mini-batch gradient descent". This distinction is unnecessary as they differ only by mini-batch size. Using mini-batch size 1 can be erratic although it may give you a chance to jump out of a local minimum.
- **Learning rate**:
  - A too high rate can lead to divergence. A too low rate can be very slow. (Figure 4-8)



  - A *learning schedule* is a scheme to gradually decrease the learning rate as we progress instead of using a constant learning rate. (This is similar to simulated annealing.)



A comparison of the effects of batch size:



Technical Notes on GD:

- Criterion for convergence: $\|\nabla C\| < \epsilon$, where $\epsilon > 0$ is very small.
- GD is a first-order approximation. In contrast, the Hessian approach is a second-order approximation (e.g., Newton–Raphson and Gauss–Newton methods).
- GD cannot guarantee to converge to the global minimum of $C(u)$. (Global minimum is guaranteed if C is convex and ∇C is Lipschitz.)

Overfitting and underfitting for models trained iteratively:

- Overfitting can be detected by monitoring the performance of the model on training and validation sets at the end of every epoch. Once the performance becomes significantly worse on the validation set, it is a sign of overfitting.
  - This motivated a regularization method called "early stopping".
- Parallel "learning curves" (i.e., performance lines for training and validation sets) can indicate a well fitted model or underfitting. This pattern is not "typical" for an underfitting model, as claimed by the author.
- Misleading statement: "One way to improve an overfitting model is to feed it more training data until ..." Unfortunately in practice, n is often fixed, and overfitting can be an issue when a model is too flexible. If n can be increased without limit, overfitting often would not be a concern.
- Traditional model diagnostic techniques can help identify underfitting when p is small.

The logistic function is $p = f(t) = \frac{e^t}{1+e^t}$. The logit function is $t = g(p) = \log(\frac{p}{1-p})$. They are the inverse of each other, not the same. Logistic regression should not be called logit regression.

6.1.4 Assignment

1. Reading for next lecture: ISLR 7
2. ISLR Chapter 7 R Labs

6.2 Week 6 Day 2

In ISLR Chapter 7, we go beyond linearity and model the nonlinear effect of a predictor. Some choices (e.g., polynomials, step functions) may not be good. Good choices are natural splines, smoothing splines, and local regression. The effects across predictors can be put together additively as in generalized additive models (GAMs).

High-dimensional versions of splines and local regression can also be defined over two or more predictors.

6.2.1 ISLR 7.1 Polynomial regression

In R, `poly(x, d)` can generate a d-degree polynomial for variable x as an $n \times d$ matrix. The default is to generate orthogonal columns, which is good for fitting models on them. To generate columns $x, x^2, \ldots, x^d$, use `poly(x, d, raw=T)`.

Polynomial regression with a high degree (> 4) is generally not recommended, because

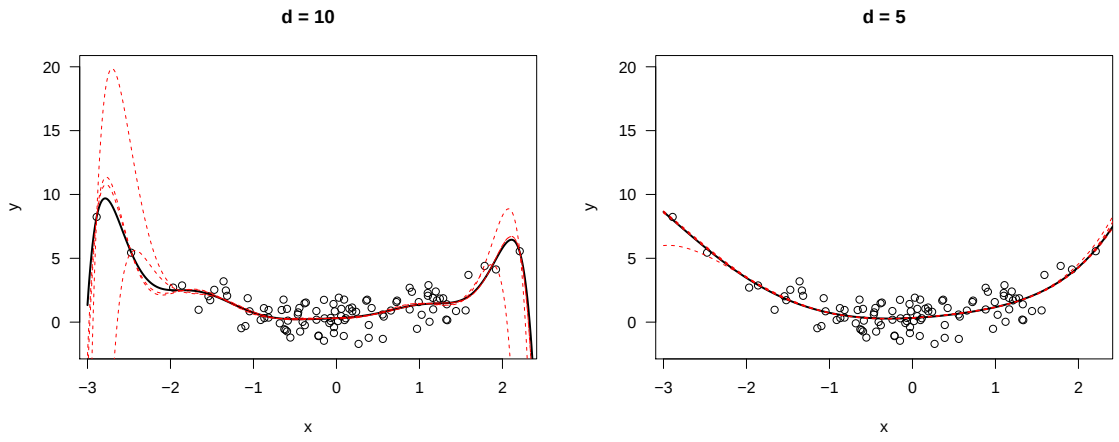
- A high-degree polynomial is difficult to interpret.
- A polynomial with K degrees has the same degrees of freedom as a natural spline with $K + 1$ knots, but the latter has a lot better flexibility and interpretation (ISLR 7.4).
- Polynomials allow high leverage data to have too much impact on the result.
- Data at one end could influence the shape of the fitted curve on the other end.

Below is an example to demonstrate the last two points.

```
set.seed(20)
n = 100
x = rnorm(n)        ## simulated x
y = x^2 + rnorm(n)  ## simulated y
xx = seq(-3, 3, 0.01)  ## a grid of x for drawing fitted curves

myfun = function(x, y, d=2, nends=5) {
    mod5 = lm(y ~ poly(x, d))
    plot(x, y, ylim=c(-2, 20), main=paste("d =", d), las=1)
    points(xx, predict(mod5, data.frame(x=xx)), type='l', lwd=2)  ## curve fitted with all obs
    orderidx = order(x)
    endpoints = c(orderidx[1:nends], tail(orderidx, nends))  ## pos of extreme x values
    for(i in endpoints) {
        xtmp = x[-i]; ytmp = y[-i]
        predtmp = predict(lm(ytmp ~ poly(xtmp, d)), data.frame(xtmp=xx))
        points(xx, predtmp, type='l', lty=2, col=2)  ## curve fitted w. an extreme value removed
    }
}

par(mfrow=c(1,2))
myfun(x, y, 10, 2)
myfun(x, y, 5, 2)
```



6.2.2 ISLR 7.2 Step functions

- Using a step function on a predictor results in a piecewise constant model.
- A step function transforms a quantitative predictor into an ordered categorical variable.
  - Common in epidemiology. For example, age group (instead of age) is often used as an input variable.
  - This can lead to distortion and loss of information. For example, when using age group (by decade) as a predictor, we implicitly assume that ages 41 and 49 (8 years apart) have the same effect, while ages 49 and 51 (2 years apart) have different effects.
  - Assuming a constant effect over an interval we may miss a trend inside the interval (Figure 7.2 shows an example.)

  - Categorization may be useful when reporting results. But not helpful in analysis.
  - Dichotomization is the worst case.
- Step functions are "easy to interpret" due to oversimplification.
  - In general, it is always "easy to interpret" if something is turned into black or white or if an effect is boiled down to a single number.
- Step functions allow nonlinear and non-monotonic effects (e.g., U-shape).

In R, `cut()` can create a factor variable from an input variable. For example:

```
x = rnorm(100)
y = x^2 + rnorm(100)
lm(y ~ cut(x, breaks=seq(-4,4)))
```

6.2.3 ISLR 7.3 Basis functions

The **basis vectors** for a space are a set of vectors that serve as the basis to span the whole space through linear combination. For example, R^3, the 3-dimensional Euclidean space has basis vectors $e_1 = (1, 0, 0)$, $e_2 = (0, 1, 0)$, $e_3 = (0, 0, 1)$. Every point in R^3 can be written as a linear combination of them: $(x, y, z) = xe_1 + ye_2 + ze_3$. They are not unique. For example $f_1 = (\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}, 0)$, $f_2 = (-\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}, 0)$, and $f_3 = (0, 0, 1)$ are another set of basis vectors.

Similarly, **basis functions** for a family of functions are a set of functions that span the whole family. For example,

- Polynomials of degree d have basis functions $f_0(x) = 1, f_1(x) = x, \ldots, f_d(x) = x^d$, because $a_0+a_1x+\cdots+a_dx^d = a_0f_0(x) + a_1f_1(x) + \cdots + a_df_d(x)$.
- Step functions over k intervals $(a_0, a_1), (a_1, a_2), \ldots, (a_{k-1}, a_k)$ have basis functions $f_1(x) = I(a_0 < x < a_1)$, $\ldots$, $f_k(x) = I(a_{k-1} < x < a_k)$.
- Basis functions are not unique. There can be several sets of basis functions.

6.2.4 ISLR 7.4 Regression splines

Splines are piecewise polynomials, with constraints at the **knots** to ensure the function is smooth.

- The constraints at a knot t are: f is continuous at t, first derivative of f is continuous at t, and second derivative of f is continuous at t. (Rationale shown in ESL Figure 5.2)

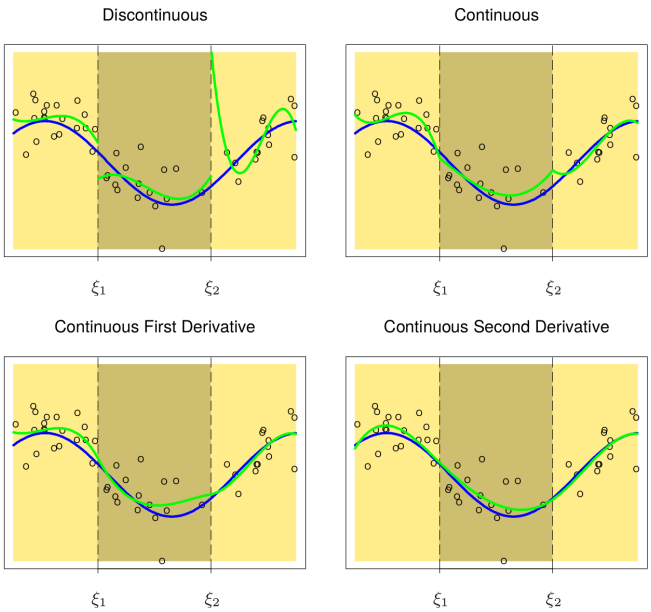


FIGURE 5.2. *A series of piecewise-cubic polynomials, with increasing orders of continuity.*

Cubic splines are splines that are piecewise cubic.

- A cubic spline with K knots has $K+1$ intervals. Each interval has 4 parameters (because a cubic function has 4 coefficients). There are 3 constraints at every knot. So, there are $4(K+1)-3K = K+4$ degrees of freedom, including one for the intercept. That is, $K+3$ DF are attributable to the variable.
- A set of basis functions are: 1, x, x^2, x^3, $(x-t_1)^3_+$, $\ldots$, $(x-t_K)^3_+$, where $t_1 < \ldots < t_K$ are the knots.
- B-splines are another set of basis functions (output of `splines::bs()`)
- In R function `splines::bs()`:
  - One can use `df=` to specify the DF attributable to the variable. For example, `bs(x, df=5)` will generate the 5 bases attributable to x (as an $n \times 5$ matrix).
  - In `bs(x, df)`, by default, the $df-3$ knots are evenly spaced according to quantile. For example, `bs(x, df=5)` will have knots at the 33th and 66th percentiles.
  - One can instead specify the positions of the knots with the `knots=` argument.

```
library(splines)  ## for bs()
attach(ISLR::Auto)

bs.mod1 = lm(mpg ~ bs(horsepower, df=5))
summary(bs.mod1)$coef
bs.mod2 = lm(mpg ~ bs(horsepower, knots=quantile(horsepower, probs=1:2/3)))
all.equal(summary(bs.mod1)$coef, summary(bs.mod2)$coef, check.attributes=F)  ## True

plot(horsepower, mpg, bty='n')
idx = order(horsepower)
points(horsepower[idx], fitted(bs.mod1)[idx], type='l', col=1, lwd=1)
detach()
```

Natural splines (NS; also called **restricted cubic splines**, RCS) are cubic splines with linear functions at the two end intervals.

- A natural spline with k knots has 2 fewer parameters at each end interval than cubic splines. So, there are $4(K-1)+2\cdot 2-3K = K$ degrees of freedom, including one for the intercept. That is, $K-1$ DF are attributable to the variable.
- A set of basis functions are $N_1(x) = 1$, $N_2(x) = x$, and for $j = 1, \ldots, K-2$,

$$N_{j+2}(x) = (x-t_j)^3_+ - \frac{t_K - t_j}{t_K - t_{K-1}}(x-t_{K-1})^3_+ + \frac{t_{K-1}-t_j}{t_K-t_{K-1}}(x-t_K)^3_+.$$

- B-splines are another set of basis functions (output of `splines::ns()`)
- In R function `splines::ns()`:
  - One can use `df=` to specify the DF attributable to the variable. For example, `ns(x, df=5)` will generate the 5 bases attributable to x (as an $n \times 5$ matrix).
  - In `ns(x, df)`, by default, the knots are $\min(x)$, $\max(x)$, and $df-1$ knots in between that are evenly spaced according to quantile. For example, `ns(x, df=5)` will have knots $\min(x)$, $\max(x)$, and 4 *internal knots* at the 20th, 40th, 60th, 80th percentiles.
  - One can instead specify the positions of the internal knots with the `knots=` argument.

```
library(splines)  ## for ns()
attach(ISLR::Auto)

ns.mod1 = lm(mpg ~ ns(horsepower, df=5))
summary(ns.mod1)$coef
ns.mod2 = lm(mpg ~ ns(horsepower, knots=quantile(horsepower, probs=1:4/5)))
all.equal(summary(ns.mod1)$coef, summary(ns.mod2)$coef, check.attributes=F)  ## True
```

```
plot(horsepower, mpg, bty='n')
idx = order(horsepower)
points(horsepower[idx], fitted(ns.mod1)[idx], type='l', col=2, lwd=1)
detach()
```

Knot positions and number of knots:

- Ideally, knots should be placed at where the function may change rapidly.
- The number of knots is a hyperparameter, which can be selected through cross-validation, as shown in Figure 7.6.

6.2.5 Assignment

1. **Homework**: ISLR Chapter 7 Exercises 9
2. Reading for next lecture: ISLR 7.5–7.7
3. ISLR Chapter 7 R Labs