STATS 415 Lab9

Weijing Tang

Mar 15, 2018

1 Today's objectives

1. Learn how to implement polynomial regression.
2. Learn how to implement regression spline and smoothing spline.
3. Learn how to implement GAM.
4. Compare the model performance among polynomial regression, splines and GAM.

2 Non-Linear Methods.

So far we have focused on linear models. They have a significant advantage: simple. It is great for interpretation and inference. In this lab, we will implement several non-linear methods whose goal is to relax the linearity assumption but attempt to maintain as much interpretability as possible.

We will use the `Wage` dataset from the `ISLR` package to explore the following three methods:

- Polynomial regression
- Splines
- Generalized additive models(GAM)

This dataset contains wage and other data for a group of 3000 male workers. We want to predict `wage` here.

```
library(ISLR)
attach(Wage)
```

3 Spliting training and test data

We will just use training data to select the relevent "degree of freedom" parameter for each model, then compare their performance on test data.

```
set.seed(1)
train = sample(1:nrow(Wage),trunc(nrow(Wage)*0.8))
```

4 Polynomial Regression

We extent linear regression to

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_d x^d + \epsilon$$

where d decides the degree of freedom of the model.

We will ues `age` to predict `wage`. In order to predict wage using a d-degree polynomial in age, we need create a matrix which contains $[x, x^1, \cdots, x^d]$ for each observation. Then we can use `lm()` function to fit a linear model as what we learned in the previous labs. `poly()` returns a matrix whose columns are a basis of orthogonal polynomials, which essentially means that each column is a linear orthogonal combination of the variables $[x, x^1, \cdots, x^d]$. It is more numerically stable compared with the original matrix.

```r
fit=lm(wage~poly(age,4),data=Wage[train,])
coef(summary(fit))
```

```
##                  Estimate Std. Error    t value     Pr(>|t|)
## (Intercept)     112.03542  0.8220573 136.286635 0.000000e+00
## poly(age, 4)1   391.65629 40.2724169   9.725175 5.963594e-22
## poly(age, 4)2  -464.46792 40.2724169 -11.533152 5.476450e-30
## poly(age, 4)3   112.27676 40.2724169   2.787932 5.346449e-03
## poly(age, 4)4   -71.15018 40.2724169  -1.766722 7.740207e-02
```

However, we can also use `poly()` to obtain [`age`,`age^2`,`age^3`,`age^4`] directly by adding `raw=TRUE` argument. Then the estimated coefficients has a clear interpretation. Later we will show that this change does not affect the fitted model (give the same prediction).

```r
fit2=lm(wage~poly(age,4,raw=T),data=Wage[train,])
coef(summary(fit2))
```

```
##                                Estimate   Std. Error   t value     Pr(>|t|)
## (Intercept)               -1.891721e+02 6.674835e+01 -2.834109 0.0046338658
## poly(age, 4, raw = T)1     2.157842e+01 6.549387e+00  3.294723 0.0009995429
## poly(age, 4, raw = T)2    -5.692226e-01 2.292830e-01 -2.482621 0.0131100098
## poly(age, 4, raw = T)3     6.837459e-03 3.408339e-03  2.006097 0.0449581428
## poly(age, 4, raw = T)4    -3.219978e-05 1.822571e-05 -1.766722 0.0774020703
```

It is equivalent to the following model:

```r
fit2a=lm(wage~age+I(age^2)+I(age^3)+I(age^4),data=Wage[train,])
coef(fit2a)
```

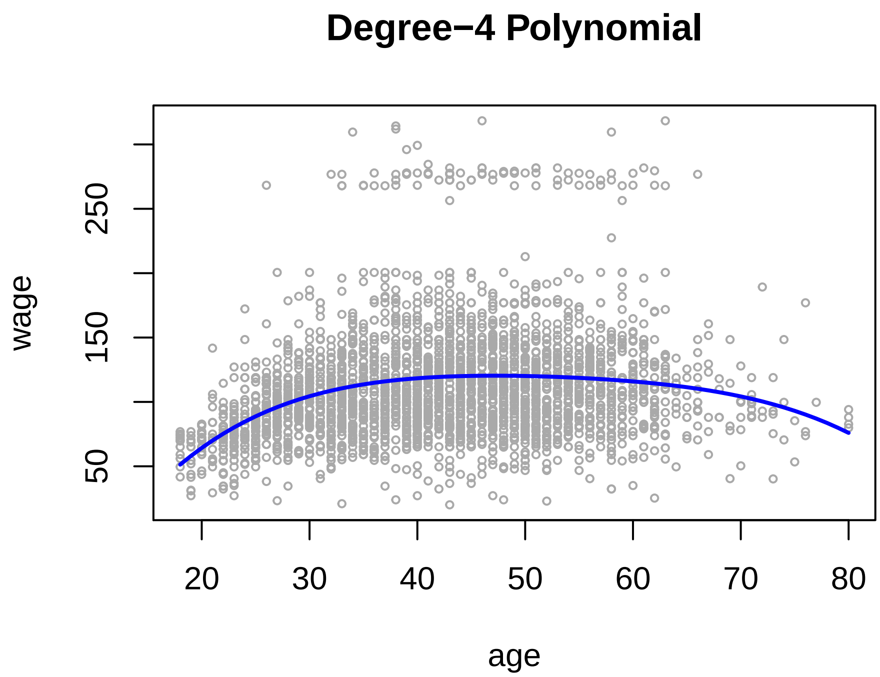
```
##   (Intercept)           age      I(age^2)      I(age^3)      I(age^4)
## -1.891721e+02  2.157842e+01 -5.692226e-01  6.837459e-03 -3.219978e-05
```

You can see the coefficient estimators are the same.

4.1 visualization

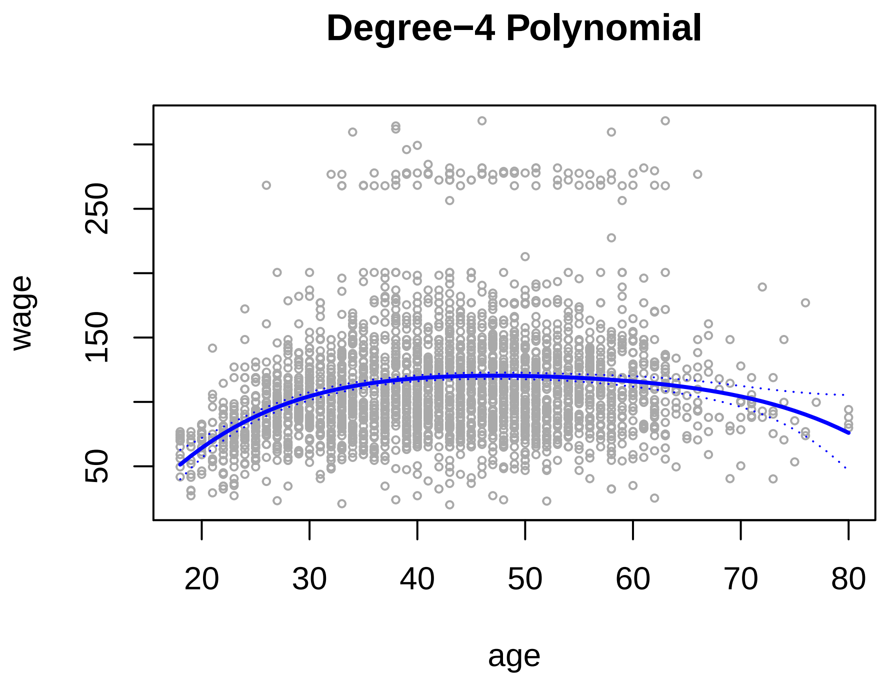
To visualize the fitted curve, we create a grid of values for `age` at which we make predictions by using generic `predict()` function.

```r
agelims=range(age)
age.grid=seq(from=agelims[1],to=agelims[2])
preds=predict(fit,newdata=data.frame(age=age.grid))
plot(age,wage,xlim=agelims,cex=.5,col="darkgrey")
title("Degree-4 Polynomial")
lines(age.grid,preds,lwd=2,col="blue")
```



We can also include standard errors in the plot by using `se=TRUE` argument of `predict()` function. Here we choose 95% confidence interval, so the band should be within roughly 1.96 standard errors of predicted values.

```
preds=predict(fit,newdata=data.frame(age=age.grid),se=TRUE)
se.bands=cbind(preds$fit+1.96*preds$se.fit,preds$fit-1.96*preds$se.fit)
plot(age,wage,xlim=agelims,cex=.5,col="darkgrey")
title("Degree-4 Polynomial")
lines(age.grid,preds$fit,lwd=2,col="blue")
lines(age.grid,se.bands[,1],lwd=1,col="blue",lty=3)
lines(age.grid,se.bands[,2],lwd=1,col="blue",lty=3)
```



We mentioned earlier that whether or not an orthogonal set of basis functions is produced in the `poly()` function will not affect the fitted model. It means the fitted values obtained in either case are identical:

```
preds2=predict(fit2,newdata=data.frame(age=age.grid),se=TRUE)
max(abs(preds$fit-preds2$fit))
```

```
## [1] 5.151435e-11
```

4.2 How to select the degree d?

In performing a polynomial regression we must decide on the degree of the polynomial to use. We will implement two methods. One way is to choose d by cross-validation. We have practiced it in Lab6. Let's go over it again!

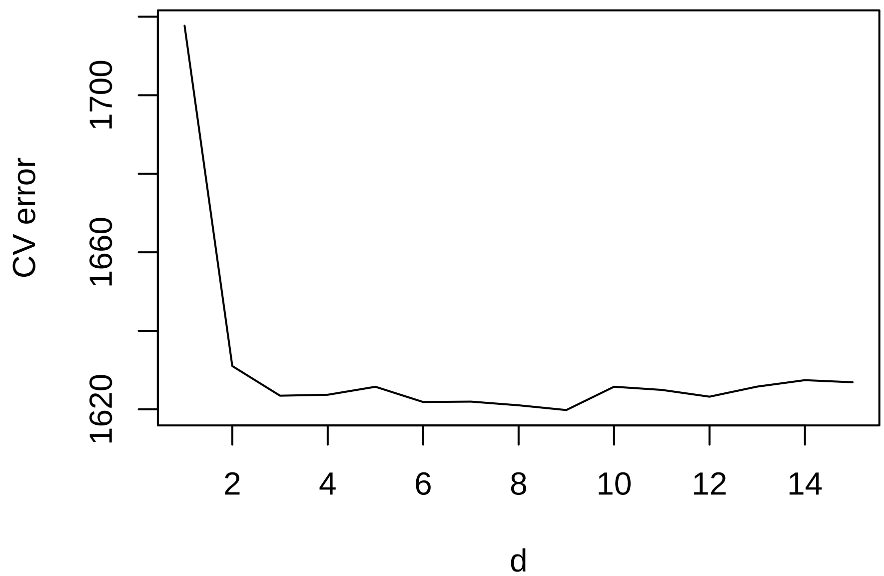
We will perform linear regression using the `glm()` function rather than the lm() function because the latter can be used together with `cv.glm()`. `cv.glm()` function is in package `boot`. We usually use 10-fold cross-validation by setting `K=10` argument. The standard cross-validation error is saved as `delta[1]` component in the retured list.

```
library(boot)
set.seed(1)
cv.error_poly = rep(0,15)
for (i in 1:15){
  fit=glm(wage~poly(age,i),data=Wage[train,])
  cv.error_poly[i]=cv.glm(Wage[train,], fit, K=10)$delta[1]
}
cv.error_poly
```

```
##  [1] 1717.707 1631.020 1623.470 1623.707 1625.740 1621.858 1621.958
##  [8] 1621.026 1619.813 1625.750 1624.945 1623.216 1625.782 1627.430
```

```
## [15] 1626.887
```

```
plot(1:15,cv.error_poly,xlab = "d",ylab = "CV error",type = "l")
```



Question: Which degree will you choose based on the result of cross-validation?

Another way is to choose d by using hypothesis tests.

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_d x^d + \epsilon$$

For example, the null hypothesis is $H_0 : \beta_{k+1} = \cdots = \beta_d = 0$. If we reject the null hypothesis, we think the polynomial regression with degree k is sufficient to explain the data against a more complex model. In general we use `anova()` function to test the null hypothesis that a model M_1 is sufficient to explain the data against the alternative hypothesis that a more complex model M_2 is required. M_1 and M_2 should be nested models if we use `anova()` function, which means that the predictors in M_1 must be a subset of the predictors in M_2. In this case, we fit models ranging from linear to a degree-5 polynomial and sequentially compare the simpler model to the more complex model, determine the simplest model which is sufficient to explain the relationship between `wage` and `age`.

```
fit.1=lm(wage~age,data=Wage[train,])
fit.2=lm(wage~poly(age,2),data=Wage[train,])
fit.3=lm(wage~poly(age,3),data=Wage[train,])
fit.4=lm(wage~poly(age,4),data=Wage[train,])
fit.5=lm(wage~poly(age,5),data=Wage[train,])
anova(fit.1,fit.2,fit.3,fit.4,fit.5)
```

```
## Analysis of Variance Table
##
## Model 1: wage ~ age
## Model 2: wage ~ poly(age, 2)
## Model 3: wage ~ poly(age, 3)
## Model 4: wage ~ poly(age, 4)
## Model 5: wage ~ poly(age, 5)
##   Res.Df     RSS Df Sum of Sq        F    Pr(>F)
## 1   2398 4117772
## 2   2397 3902041  1    215730 133.0405 < 2.2e-16 ***
## 3   2396 3889435  1     12606   7.7741  0.005342 **
## 4   2395 3884373  1      5062   3.1219  0.077372 .
## 5   2394 3881967  1      2406   1.4838  0.223294
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

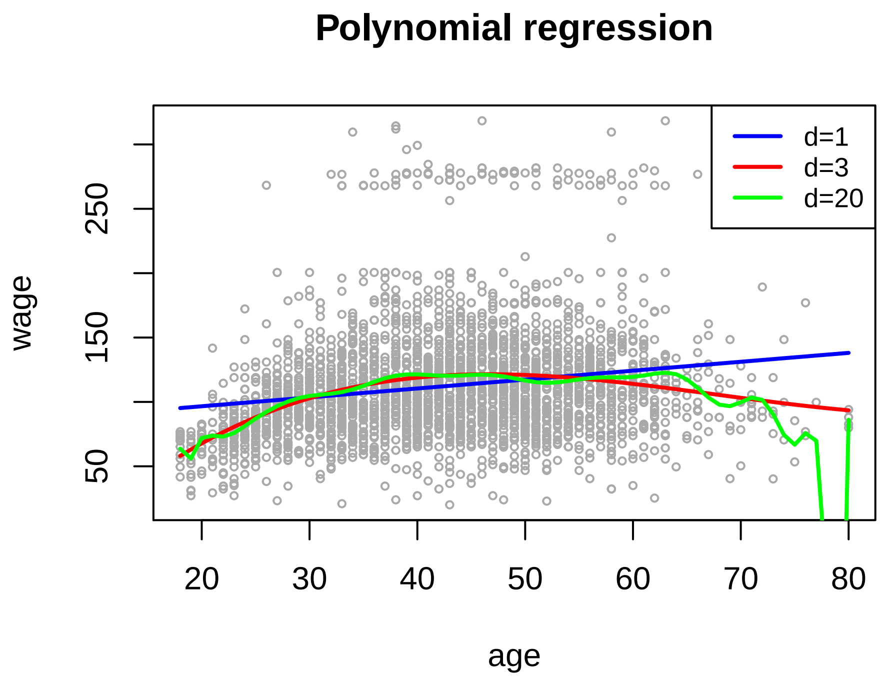
Question: Which degree will you choose based on the result of anova(or p-values)? Which degree will you choose finally? 3

We prefer to choose 3. Because after 3 the error is quite similar and the model for 3 is simpler

4.3 What kind of role does degree d play in the model?

Let's visualize the fitted curve with different degrees in one plot!

```
fit.20=lm(wage~poly(age,20),data=Wage[train,])
preds1=predict(fit.1,newdata=data.frame(age=age.grid))
preds3=predict(fit.3,newdata=data.frame(age=age.grid))
preds20=predict(fit.20,newdata=data.frame(age=age.grid))
plot(age,wage,xlim=agelims,cex=.5,col="darkgrey")
title("Polynomial regression")
lines(age.grid,preds1,lwd=2,col="blue")
lines(age.grid,preds3,lwd=2,col="red")
lines(age.grid,preds20,lwd=2,col="green")
legend("topright",legend=c("d=1","d=3","d=20"),col=c("blue","red","green"),lty=1,lwd=2,cex=.8)
```




```
fit.poly = glm(wage~poly(age,3),data=Wage[train,])
```

Question: How does the fitted curve change as we increase the degree d?

overfit

5 Splines

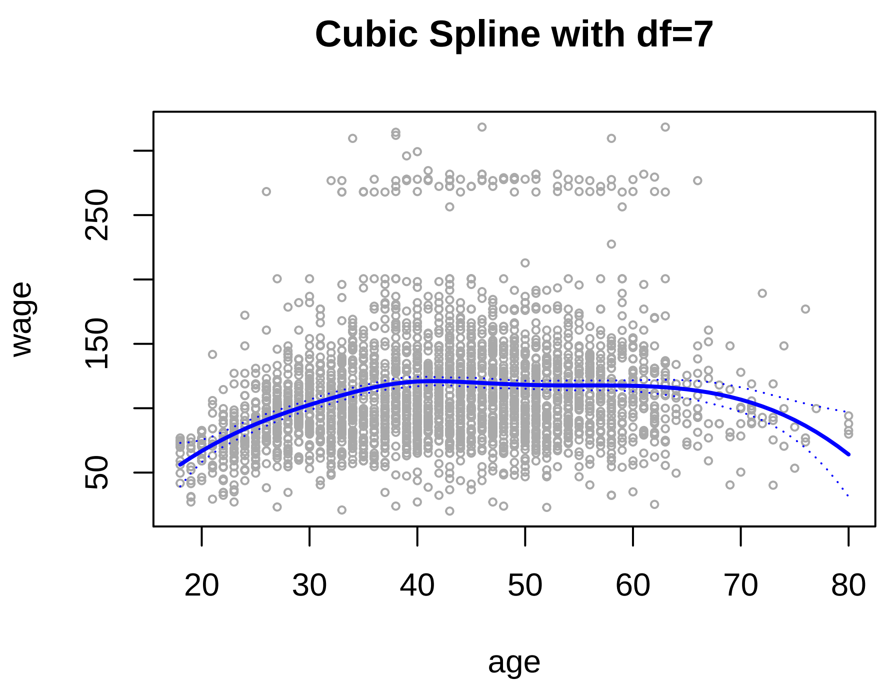
The key idea of regression splines is to fit different polynomials locally(over different regions of x). Fortunately we can represent a spline with k knots by constructing an appropriate matrix of basis functions.

5.1 Cubic splines

The `bs()` function generates the entire matrix for splines with the specified set of knots, which is in package `splines`. The degree of freedom is decided by the degree of polynomial and the number of knots. We can set `degree=d` and `knots=...` or set `df=` directly. By default, `degree=3` for cubic splines. In the case of cubic spline, $df = \#knots + 4$. In practice, knots are often placed at uniform quantiles of the data. For example, if we just set `df=7`, `bs()` function will generate the matrix for splines with knots placed at 0.25, 0.5, 0.75 quantiles of `age`. This produces a spline with 7 basis functions.(including the intercept term)

```
library(splines)
fit=lm(wage~bs(age,df = 7),data=Wage[train,])
```

```r
preds=predict(fit,newdata=data.frame(age=age.grid),se=TRUE)
se.bands=cbind(preds$fit+1.96*preds$se.fit,preds$fit-1.96*preds$se.fit)
plot(age,wage,xlim=agelims,cex=.5,col="darkgrey")
title("Cubic Spline with df=7")
lines(age.grid,preds$fit,lwd=2,col="blue")
lines(age.grid,se.bands[,1],lwd=1,col="blue",lty=3)
lines(age.grid,se.bands[,2],lwd=1,col="blue",lty=3)
```



We can also use specified knots.

```r
dim(bs(age,knots=c(25,40,60)))
```

```
## [1] 3000    6
```

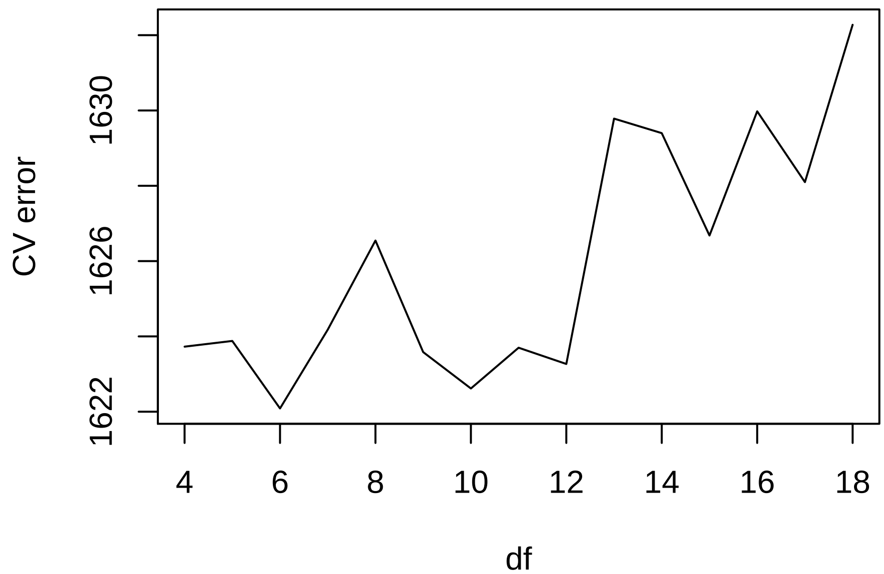
5.2 How to choose the degree of freedom of splines?

We can choose df by cross-validation. degree!=df

```r
set.seed(1)
cv.error_cs = rep(0,15)
for (i in 1:15){
  fit=glm(wage~bs(age,df = i+3),data=Wage[train,])
  cv.error_cs[i]=cv.glm(Wage[train,], fit, K=10)$delta[1]
}
which.min(cv.error_cs)
```

```
## [1] 3
```

```r
plot(4:18,cv.error_cs,xlab = "df",ylab = "CV error",type = "l")
```




```
fit.cs = glm(wage~bs(age,df = 6),data=Wage[train,])
```

Question: How many knots will you choose based on the result of cross-validation?

5.3 Natural splines

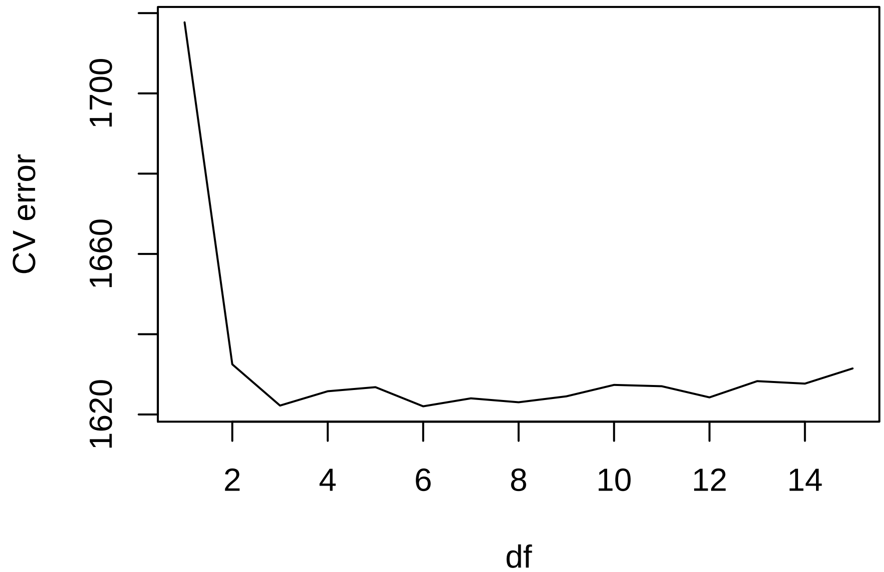
Natural spline replaces the "end" cubic splines (one on each side) with lines which gains stability on the boundary. Refer to the plot on Page 24 in the lecture notes. We use `ns()` function instead of `bs()`. `ns()` function works similarly as `bs()`. In the case of natural spline, $df = \#knots + 1$.

Let's choose the degree of freedom again by cross-validation.

```
set.seed(1)
cv.error_ns = rep(0,15)
for (i in 1:15){
  fit=glm(wage~ns(age,df = i),data=Wage[train,])
  cv.error_ns[i]=cv.glm(Wage[train,], fit, K=10)$delta[1]
}
which.min(cv.error_ns)
```

```
## [1] 6
```

```r
plot(1:15,cv.error_ns,xlab = "df",ylab = "CV error",type = "l")
```




```r
fit.ns = glm(wage~ns(age,df = 6),data=Wage[train,])
```

5.4 Smoothing spline

The key idea of smooth splines is to find some function g that makes RSS small but is also smooth. The objective function is

$$\sum_{i=1}^{n}(y_i - g(x_i))^2 + \lambda \int g''(t)^2 dt$$

where the smoothness of g is controlled by tuning parameter λ.

Question: What kind of model we are fitting if $\lambda = \infty$?

We don't give a explicit form of the degree of freedom, but it is uniquely defined by λ. In general, the larger λ is, the smaller the degree of freedom is. We use the `smooth.spline()` function to fit a smoothing spline.

```r
fit=smooth.spline(age,wage,df=16)
```

We can do cross-validation and visualization by using `smooth.spline()` function directly.

```
fit.ss=smooth.spline(age,wage,cv=TRUE)
```

```
## Warning in smooth.spline(age, wage, cv = TRUE): cross-validation with non-
## unique 'x' values seems doubtful
```

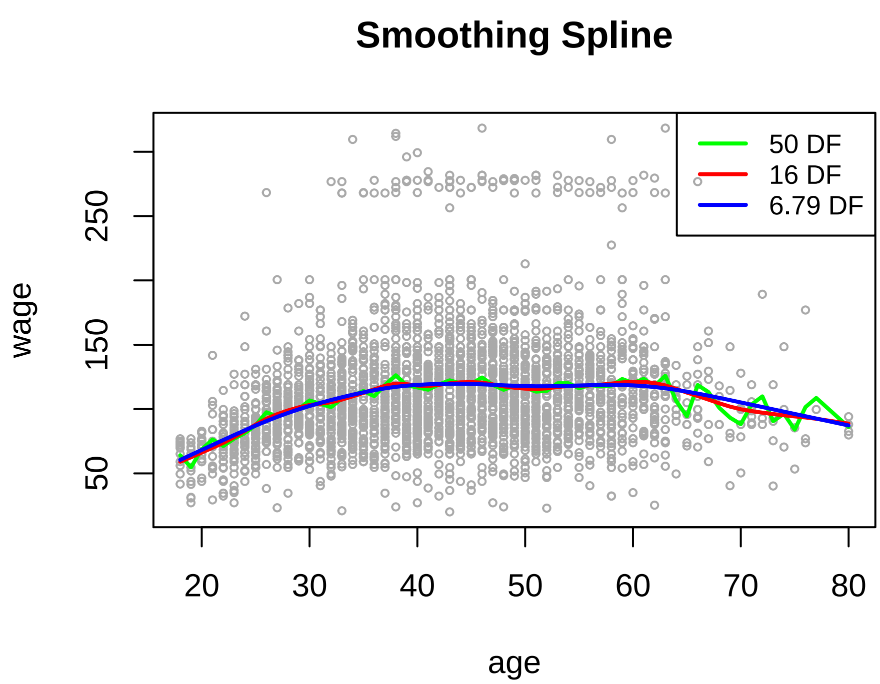
```
fit.ss$df
```

```
## [1] 6.794596
```

```
fit.ss$lambda
```

```
## [1] 0.02792303
```

```
fit.50=smooth.spline(age,wage,df=50)
plot(age,wage,xlim=agelims,cex=.5,col="darkgrey")
title("Smoothing Spline")
lines(fit.50,col="green",lwd=2)
lines(fit,col="red",lwd=2)
lines(fit.ss,col="blue",lwd=2)
legend("topright",legend=c("50 DF","16 DF","6.79 DF"),col=c("green","red","blue"),lty=1,lwd=2,cex=.8)
```



Question: How does the fitted curve change as we increase the degree of freedom?

6 GAMs

GAM replaces each term in the multiple linear regression model with a non-linear function:

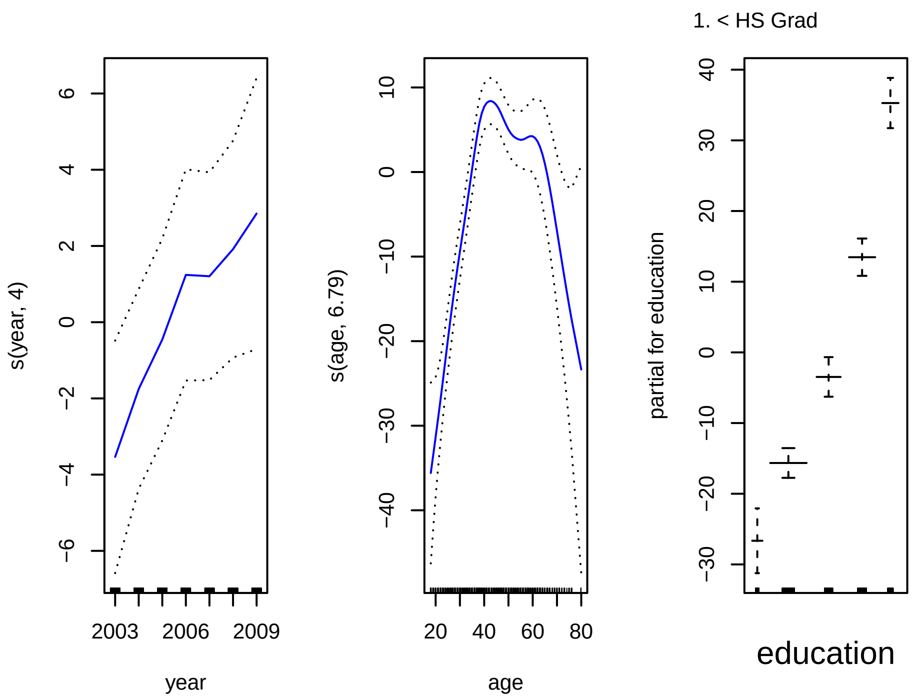
$$y = \beta_0 + f_1(x_1) + f_2(x_2) + \cdots + f_p(x_p) + \epsilon$$

We now fit a GAM to predict wage using natural spline functions of year and age, treating education as a qualitative predictor. We can create spline matrix of basis functions by using `ns()`, `bs()` functions and use `lm()` function.

```
gam1=lm(wage~ns(year,4)+ns(age,5)+education,data=Wage[train,])
```

In order to fit more general sorts of GAMs, we will need to use the `gam` library in R. The `s()` function, which is part of the `gam` library, is used to indicate that we would like to use a smoothing spline. The second argument is the target degree of freedom. Here we use smoothing spline with df $= 6.79$ for predictor `age`. Let's also try smoothing spline for `year`.

```
library(gam)
gam2=gam(wage~s(year,4)+s(age,6.79)+education,data=Wage[train,])
par(mfrow=c(1,3))
plot(gam2, se=TRUE,col="blue")
```



Notice that if we want to plot a GAM fitted object obtained from `lm()`, we had to use `plot.gam()` rather than the generic `plot()` function.

In these plots, the function of `year` looks rather linear. We can perform ANOVA test in order to determine which of the models is better: a GAM that uses a linear function of year (M_1), or a GAM that uses a spline function of year (M_2).

```
gam.m1=gam(wage~year+s(age,6.79)+education,data=Wage[train,])
gam.m2=gam(wage~s(year,4)+s(age,6.79)+education,data=Wage[train,])
anova(gam.m1,gam.m2,test="F")
```

```
## Analysis of Deviance Table
##
## Model 1: wage ~ year + s(age, 6.79) + education
## Model 2: wage ~ s(year, 4) + s(age, 6.79) + education
##   Resid. Df Resid. Dev Df Deviance      F Pr(>F)
## 1    2387.2    3004854
## 2    2384.2    3003602  3   1251.4 0.3311 0.8029
```

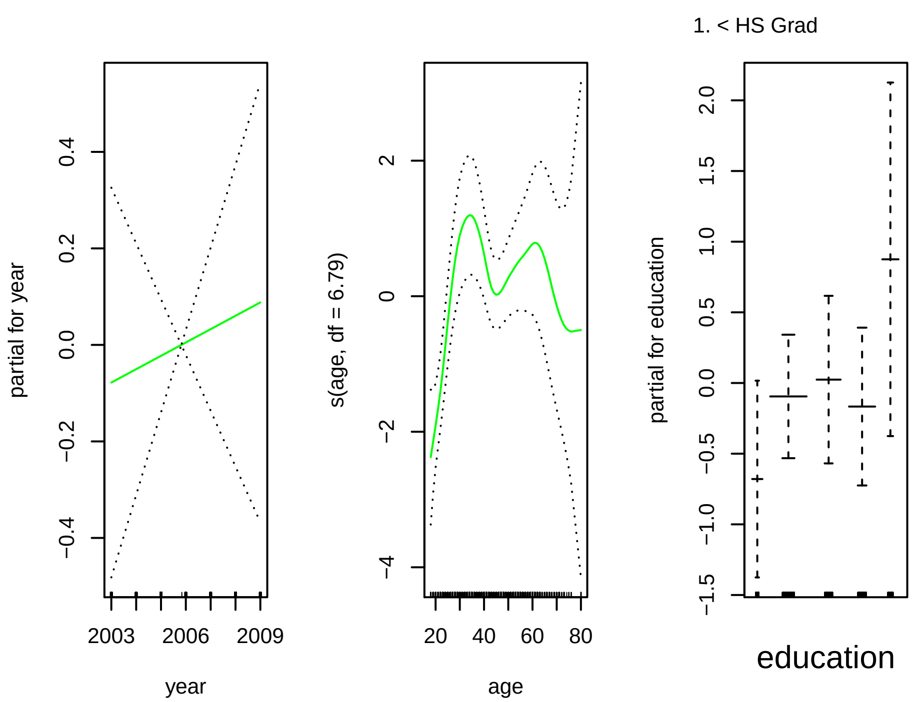
Question: Based on the p-value in the anova result, which model is preferred?

We can make predictions from `gam` objects, just like from `lm` objects, using the `predict()` method for the class `gam`.

6.1 logistic GAM

We set argument `family=binomial` as what we did for logistic regression using `glm()` function. In order to fit a logistic regression GAM, we use the I() function in constructing the binary response variable.

```
gam.lr=gam(I(wage>50) ~ year + s(age,df=6.79) + education, family=binomial,data=Wage[train,])
par(mfrow=c(1,3))
plot(gam.lr,se=T,col="green")
```



7 Compare the performance of polynomial regression, cubic spline, natural spline and smoothing spline

```
test.poly = predict(fit.poly,Wage[-train,])
test.cs = predict(fit.cs,Wage[-train,])
test.ns = predict(fit.ns,Wage[-train,])
test.ss = predict(fit.ss,age[-train])
test.gam = predict(gam.m1,Wage[-train,])
wage.test= wage[-train]
error.poly = mean(wage.test-test.poly)
error.cs = mean(wage.test-test.cs)
error.ns = mean(wage.test-test.ns)
error.ss = mean(wage.test-test.ss$y)
error.gam = mean(wage.test-test.gam)
d <- data.frame("TestMSE" = c(error.poly, error.cs, error.ns, error.ss, error.gam))
rownames(d) <- c("poly regression", "cubic spline", "natural spline", "smoothing spline","GAM")
knitr::kable(d)
```

	TestMSE
poly regression	-2.275973
cubic spline	-2.392264
natural spline	-2.406441
smoothing spline	-1.893247
GAM	-1.434419

Question: Which model do you prefer from the table above?

8 Resources and References

1. Partially adapted from http://www-bcf.usc.edu/~gareth/ISL/Chapter%207%20Lab.txt