UTD 2016 Spring

CS6301.5U1 Advanced Computational Methods for Data Science

Assignment 6 – Non-linear Models (Polynomial Regression, Cubic Spline, Natural Spline and Smoothing Spline)

Name: Yu Zhang, Mingyue Sun

NetID: yxz141631, mxs151730

Part 1. Polynomial Regression

In this part we will fit the data with polynomial models with degree from 1 to 4. And we will use 10-folds cross validation to verify the test MSE to compare those models generated above. At last we chose the model with the least MSE as the model of polynomial regression.

1.1 Degree=1

```
> #Polynomial Regression (degree=1)
> k=10
> err=seq(1:10)
> folds=sample(1:k,nrow(abalone),replace = TRUE)
> for(i in 1:k)
+ {
+   train=abalone[folds!=i,]
+   test=abalone[folds==i,]
+   fit=lm(VWGHT~Diameter,data=train)
+   pred=predict(fit,test)
+   mse.temp=mean((pred-test$VWGHT)^2)
+   err[i]=mse.temp
+ }
> err
 [1] 0.002352794 0.002062566 0.002634482 0.002182009 0.002214859 0.002206191 0.002372646 0.001971080 0.002516097
[10] 0.002404852
> mse1=mean(err)
> mse1
[1] 0.002291758
```

The MSE in polynomial model with degree of 1 is 0.002291758.

1.2Degree=2

```
> #Polynomial Regression (degree=2)
> k=10
> err=seq(1:10)
> folds=sample(1:k,nrow(abalone),replace = TRUE)
> for(i in 1:k)
+ {
+   train=abalone[folds!=i,]
+   test=abalone[folds==i,]
+   fit=lm(VWGHT~Diameter+Diameter^2,data=train)
+   pred=predict(fit,test)
+   mse.temp=mean((pred-test$VWGHT)^2)
+   err[i]=mse.temp
+ }
> err
 [1] 0.002647644 0.002157021 0.002199168 0.002220790 0.002095100 0.002087478 0.002217341 0.002362035 0.002139728
[10] 0.002816193
> mse2=mean(err)
> mse2
[1] 0.00229425
```

The MSE in polynomial model with degree of 2 is 0.00229425.

1.3Degree=3

```
> #Polynomial Regression (degree=3)
> k=10
> err=seq(1:10)
> folds=sample(1:k,nrow(abalone),replace = TRUE)
> for(i in 1:k)
+ {
+   train=abalone[folds!=i,]
+   test=abalone[folds==i,]
+   fit=lm(VWGHT~Diameter+Diameter^2+Diameter^3,data=train)
+   pred=predict(fit,test)
+   mse.temp=mean((pred-test$VWGHT)^2)
+   err[i]=mse.temp
+ }
> err
 [1] 0.002632310 0.002254663 0.002198655 0.001994541 0.002056984 0.001967936 0.002652585 0.002366695 0.002492506
[10] 0.002315595
> mse3=mean(err)
> mse3
[1] 0.002293247
```

The MSE in polynomial model with degree of 3 is 0.002293247.

1.4Degree=4

```
> #Polynomial Regression (degree=4)
> k=10
> err=seq(1:10)
> folds=sample(1:k,nrow(abalone),replace = TRUE)
> for(i in 1:k)
+ {
+   train=abalone[folds!=i,]
+   test=abalone[folds==i,]
+   fit=lm(VWGHT~Diameter+Diameter^2+Diameter^3+Diameter^4,data=train)
+   pred=predict(fit,test)
+   mse.temp=mean((pred-test$VWGHT)^2)
+   err[i]=mse.temp
+ }
> err
 [1] 0.002279019 0.002113504 0.002533562 0.002431283 0.002122856 0.001892117 0.002574975 0.002081889 0.002263017
[10] 0.002591482
> mse4=mean(err)
> mse4
[1] 0.002288371
```

The MSE in polynomial model with degree of 1 is 0.002288371.

1.5Conclusion for Polynomial Model Fitting

Fro mthe analysis above we can see that when the degree equals 4 we can get the least MSE, which is 0.002288371. So we will use the polynomial model with degree of 4.

Part 2. Cubic Spline

In this part we will try to fit model with cubic spline. In implementation we will use B-spline with degree of 3 (the default setting of B-spline in R). And we will use 10-folds creoss validation to get the MSE of test sets.

```
> #Cubic Spline
> k=10
> err=seq(1:10)
> folds=sample(1:k,nrow(abalone),replace = TRUE)
> for(i in 1:k)
+ {
+   train=abalone[folds!=i,]
+   test=abalone[folds==i,]
+   fit=lm(VWGHT~bs(Diameter, knots=c(.2, .4, .5)), data=train)
+   pred=predict(fit,test)
+   mse.temp=mean((pred-test$VWGHT)^2)
+   err[i]=mse.temp
+ }
Warning messages:
1: In bs(Diameter, degree = 3L, knots = c(0.2, 0.4, 0.5), Boundary.knots = c(0.055,  :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
2: In bs(Diameter, degree = 3L, knots = c(0.2, 0.4, 0.5), Boundary.knots = c(0.09,  :
  some 'x' values beyond boundary knots may cause ill-conditioned bases
> err
 [1] 0.001647911 0.001364587 0.001419435 0.001418255 0.001406325 0.001442486 0.001754727 0.001306051 0.001583452
[10] 0.001871594
> mse.cs=mean(err)
> mse.cs
[1] 0.001521482
```

We can see the warning message above that some points may break the knots we set. The reason of this problem is that when we do sampling, in some regions we set by knots may be empty. We choose to ignore the warning here since it will not do much damage to our model.

From the result we can see that the MSE for cubic spline model is 0.001521482. This is much smaller than that of polynomial regression.

Part 3. Natural splines

For natural splines model set-up, we would like to try different degree of freedom (df) on the model to see how the test MSE changes and find the best MSE for this model. And we will perform a 10-fold CV on this model as well

First we tried df = 2

```
> #natural splines
> k=10
> folds=sample(1:k,nrow(Abalone),replace = TRUE)
> #degree of freedom = 2
> err2=seq(1:10)
> for(i in 1:k)
+ {
+   train2=Abalone[folds!=i,]
+   test2=Abalone[folds==i,]
+   fit.nspline2 = lm(VWGHT~ns(Diameter,df=2),data=Abalone)
+   pred2=predict(fit.nspline2,test2)
+   mse.temp2=mean((pred2-test2$VWGHT)^2)
+   err2[i]=mse.temp2
+ }
> err2
 [1] 0.001253467 0.001656430 0.001787527 0.001431317 0.001355523 0.001468958
 [7] 0.001480941 0.001501322 0.001720630 0.001549017
> mse2=mean(err2)
> mse2
[1] 0.001520513
```

And the result turned out to be that the test MSE for the natural splines model with df=2 is 0.001520513

Then we tried df = 4 (default value)

```
> #degree of freedom = 4
> err4=seq(1:10)
> for(i in 1:k)
+ {
+   train4=Abalone[folds!=i,]
+   test4=Abalone[folds==i,]
+   fit.nspline4 = lm(VWGHT~ns(Diameter,df=4),data=Abalone)
+   pred4=predict(fit.nspline4,test4)
+   mse.temp4=mean((pred4-test4$VWGHT)^2)
+   err4[i]=mse.temp4
+ }
> err4
 [1] 0.001252895 0.001627000 0.001769568 0.001428405 0.001370052 0.001471363
 [7] 0.001488278 0.001509081 0.001712380 0.001542028
> mse4=mean(err4)
> mse4
[1] 0.001517105
```

And the result turned out to be that the test MSE for the natural splines model with df=4 is 0.001517105

And we tried df = 6

```
> #degree of freedom = 6
> err6=seq(1:10)
> for(i in 1:k)
+ {
+   train6=Abalone[folds!=i,]
+   test6=Abalone[folds==i,]
+   fit.nspline6 = lm(VWGHT~ns(Diameter,df=6),data=Abalone)
+   pred6=predict(fit.nspline6,test6)
+   mse.temp6=mean((pred6-test6$VWGHT)^2)
+   err6[i]=mse.temp6
+ }
> err6
 [1] 0.001251844 0.001628548 0.001772195 0.001429107 0.001367833 0.001469900
 [7] 0.001487308 0.001506638 0.001713330 0.001537216
> mse6=mean(err6)
> mse6
[1] 0.001516392
>
```

And the result turned out to be that the test MSE for the natural splines model with df=6 is 0.001516392

Finally we tried df = 8

```
> #degree of freedom = 8
> err8=seq(1:10)
> for(i in 1:k)
+ {
+   train8=Abalone[folds!=i,]
+   test8=Abalone[folds==i,]
+   fit.nspline8 = lm(VWGHT~ns(Diameter,df=8),data=Abalone)
+   pred8=predict(fit.nspline8,test8)
+   mse.temp8=mean((pred8-test8$VWGHT)^2)
+   err8[i]=mse.temp8
+ }
> err8
 [1] 0.001256179 0.001602445 0.001774585 0.001427209 0.001376106 0.001457507
 [7] 0.001498541 0.001502417 0.001711733 0.001522621
> mse8=mean(err8)
> mse8
[1] 0.001512934
>
```

And the result turned out to be that the test MSE for the natural splines model with df=8 is 0.001512934

```
> min(mse2,mse4,mse6,mse8)
[1] 0.001512934
```

And we found out that the best test MSE for the natural splines model we can get by changing the df is that when df = 8, the MSE is **0.001512934**

Part4. Smoothing splines

For smoothing splines model set-up, we also would like to try different degree of freedom (df) on the model to see how the test MSE changes and find the best MSE for this model. In the same way, we will perform a 10-fold CV on this model

First we tried df = 3

```
> #degree of freedom = 3
> err3=seq(1:10)
> for(i in 1:k)
+ {
+   train3=Abalone[folds!=i,]
+   test3=Abalone[folds==i,]
+   fit.sspline3 = smooth.spline(train3$Diameter,train3$VWGHT, df=3)
+   pred_ss3=predict(fit.sspline3,test3$Diameter)
+   mse.temp3=mean((pred_ss3$y-test3$VWGHT)^2)
+   err3[i]=mse.temp3
+ }
> err3
 [1] 0.001382446 0.001865446 0.001919460 0.001565481 0.001424651 0.001555698
 [7] 0.001530494 0.001596704 0.001811656 0.001692493
> mse3=mean(err3)
> mse3
[1] 0.001634453
```

And the result turned out to be that the test MSE for the smoothing splines model with df=3 is 0.001634453

Then we tried df = 5

```
> #degree of freedom = 5
> err5=seq(1:10)
> for(i in 1:k)
+ {
+   train5=Abalone[folds!=i,]
+   test5=Abalone[folds==i,]
+   fit.sspline5 = smooth.spline(train5$Diameter,train5$VWGHT, df=5)
+   pred_ss5=predict(fit.sspline5,test5$Diameter)
+   mse.temp5=mean((pred_ss5$y-test5$VWGHT)^2)
+   err5[i]=mse.temp5
+ }
> err5
 [1] 0.001255342 0.001675691 0.001788960 0.001436677 0.001354675 0.001468034
 [7] 0.001479132 0.001506554 0.001720348 0.001553444
> mse5=mean(err5)
> mse5
[1] 0.001523885
>
```

And the result turned out to be that the test MSE for the smoothing splines model with df=5 is 0.001523885

And we tried df=7(default value)

```
> #degree of freedom = 7
> err7=seq(1:10)
> for(i in 1:k)
+ {
+   train7=Abalone[folds!=i,]
+   test7=Abalone[folds==i,]
+   fit.sspline7 = smooth.spline(train7$Diameter,train7$VWGHT, df=7)
+   pred_ss7=predict(fit.sspline7,test7$Diameter)
+   mse.temp7=mean((pred_ss7$y-test7$VWGHT)^2)
+   err7[i]=mse.temp7
+ }
> err7
 [1] 0.001252342 0.001653645 0.001778969 0.001431983 0.001367593 0.001468966
 [7] 0.001487837 0.001507766 0.001715423 0.001541331
> mse7=mean(err7)
> mse7
[1] 0.001520586
```

And the result turned out to be that the test MSE for the smoothing splines model with df=7 is 0.001520586

Finally we tried df=9

```
> #degree of freedom = 9
> err9=seq(1:10)
> for(i in 1:k)
+ {
+   train9=Abalone[folds!=i,]
+   test9=Abalone[folds==i,]
+   fit.sspline9 = smooth.spline(train9$Diameter,train9$VWGHT, df=9)
+   pred_ss9=predict(fit.sspline9,test9$Diameter)
+   mse.temp9=mean((pred_ss9$y-test9$VWGHT)^2)
+   err9[i]=mse.temp9
+ }
> err9
 [1] 0.001252463 0.001647318 0.001776781 0.001431451 0.001378605 0.001468462
 [7] 0.001493457 0.001507651 0.001715173 0.001535942
> mse9=mean(err9)
> mse9
[1] 0.00152073
```

And the result turned out to be that the test MSE for the smoothing splines model with df=9 is 0.00152073

```
> min(mse3,mse5,mse7,mse9)
[1] 0.001520586
```

And we found out that the best test MSE for the smoothing splines model we can get by changing the df is that when df = 7, the MSE is **0.001520586**

Part 5. Conclusion

Model	Polynomial Regression	Cubic Spline	Natural Spline	Smoothing Spline
MSE	**0.002291758**	**0.001521482**	**0.001512934**	**0.001520586**

From the table of comparison of MSEs above we can conclude that the Natural Spline has the lowest MSE for this dataset.

The Polynomial Regression has the largest MSE which means it's the worst model for this case. The reason behind this may be that Polynomial Regression try to fit the model from the entire dataset. However, for this "Abalone" dataset the distribution of the data points has some patterns which indicates the patterns of data distribution may vary from section to section.

Spline methods can fit the dataset well because they treat the dataset separately in different sections, which is split by knots or from degree of freedoms. So we can clearly see that the MSEs are close to each other. Cubic spline has relatively poor performance because it does not take care of the data points in boundaries (front and end). We can indicate that through

the plot below. Natural spline, however fixed the problem. So, it has a very low MSE. Smoothing spline trying to give more weight to data points that are close to current point. Therefore, it loses some accuracies in fitting training model. But in real world, the smoothing spline may have better performance.