STAT3612_2312_DATAMINING_6TH_TUTORIAL

YOU Jia

3/13/2017

Cross-Validation for Predictive Analytics

In other words, a predictive model is considered good when it is capable of predicting previously unseen samples with high accuracy. The accuracy of a model's predictions is usually gauged using a loss function. Popular choices for the loss functions are the mean-squared error for continuous outcomes, or the 0-1 loss for a categorical outcome.

- *Training error*: the average loss over the training sample.
- *Test error*: the prediction error over an independent test sample.

The training error gets smaller as long as the predicted responses are close to the observed responses, and will get larger if for some of the observations, the predicted and observed responses differ substantially. The training error is calculated using the training sample used to fit the model. Clearly, we shouldn't care too much about the model's predictive accuracy on the training data. On the contrary, we would like to assess the model's ability to predict observations never seen during estimation. The test error provides a measure of this ability. In general, one should select the model corresponding to the lowest test error.

Let's take a look at the example from previous Lecture Note

Data Generation

```
ffun = function(x) exp(-(x-3)^2)
x = seq(0.1,4,by=0.1)
set.seed(2017)
y = ffun(x) + 0.1*rnorm(length(x))
DataX = data.frame(x,y)
```

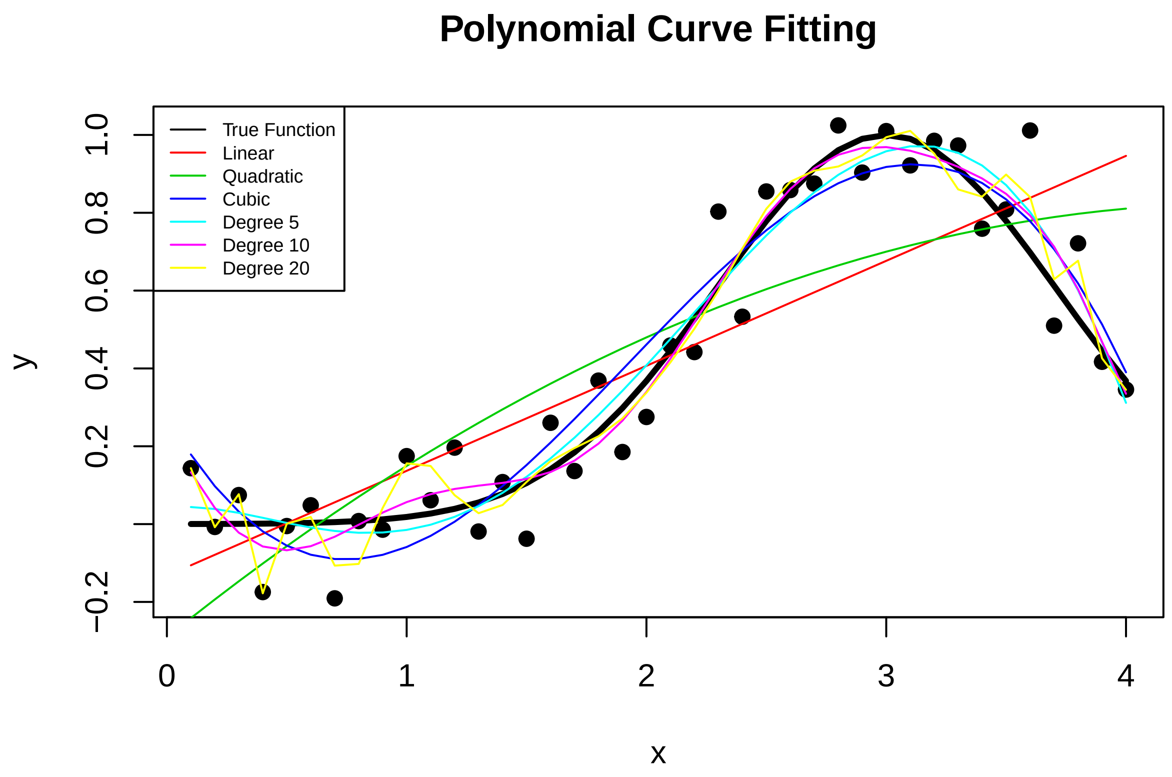
Model fitting regarding different degrees of polynomial

```
fit1 = lm(y~x) # Linear model
fit2 = lm(y ~ poly(x,2))  # Quadratic model
fit3 = lm(y ~ poly(x,3))  # Cubic model
fit4 = lm(y ~ poly(x,5))  # Degree 5
fit5 = lm(y ~ poly(x,10))  # Degree 10
fit6 = lm(y ~ poly(x,20))  # Degree 20
```

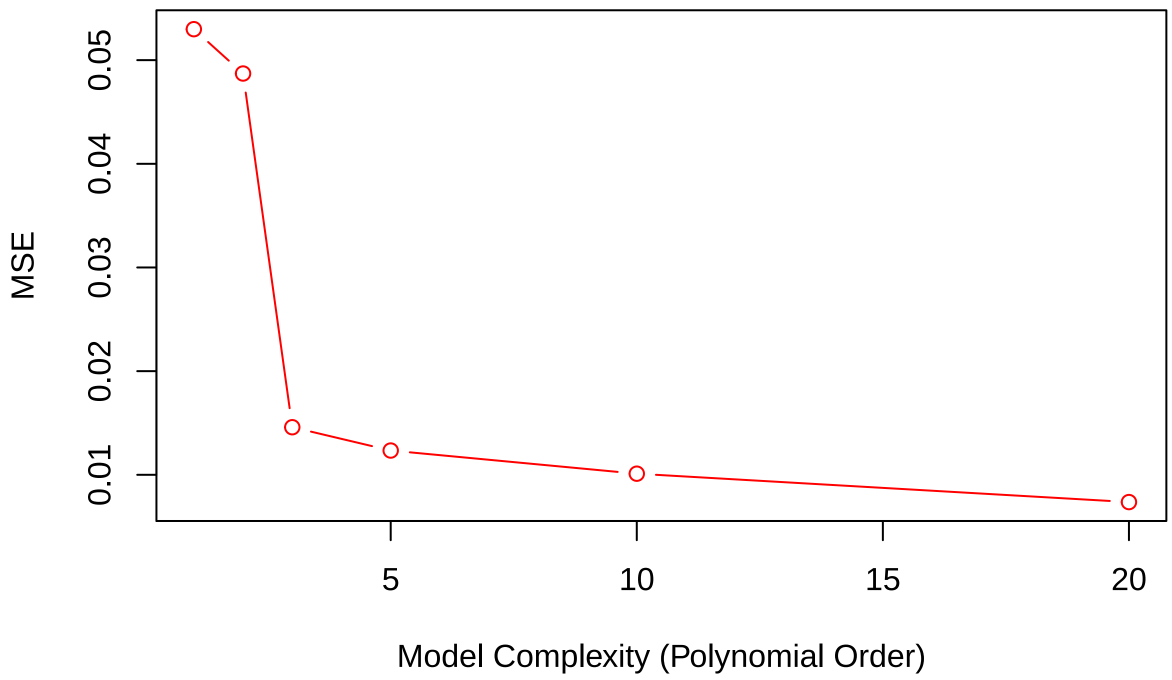
Plotting fitted models

```
plot(x, y, pch=19)
lines(x,ffun(x),col=1, lwd = 3)
lines(x, fit1$fitted.values, col=2)
lines(x, fit2$fitted.values, col=3)
lines(x, fit3$fitted.values, col=4)
lines(x, fit4$fitted.values, col=5)
```

```r
lines(x, fit5$fitted.values, col=6)
lines(x, fit6$fitted.values, col=7)
legend("topleft",
       c("True Function","Linear", "Quadratic","Cubic",
         "Degree 5","Degree 10","Degree 20"),
       lty=1,
       col=c(1,2,3,4,5,6,7),
       cex=0.6)
title(main="Polynomial Curve Fitting")
```

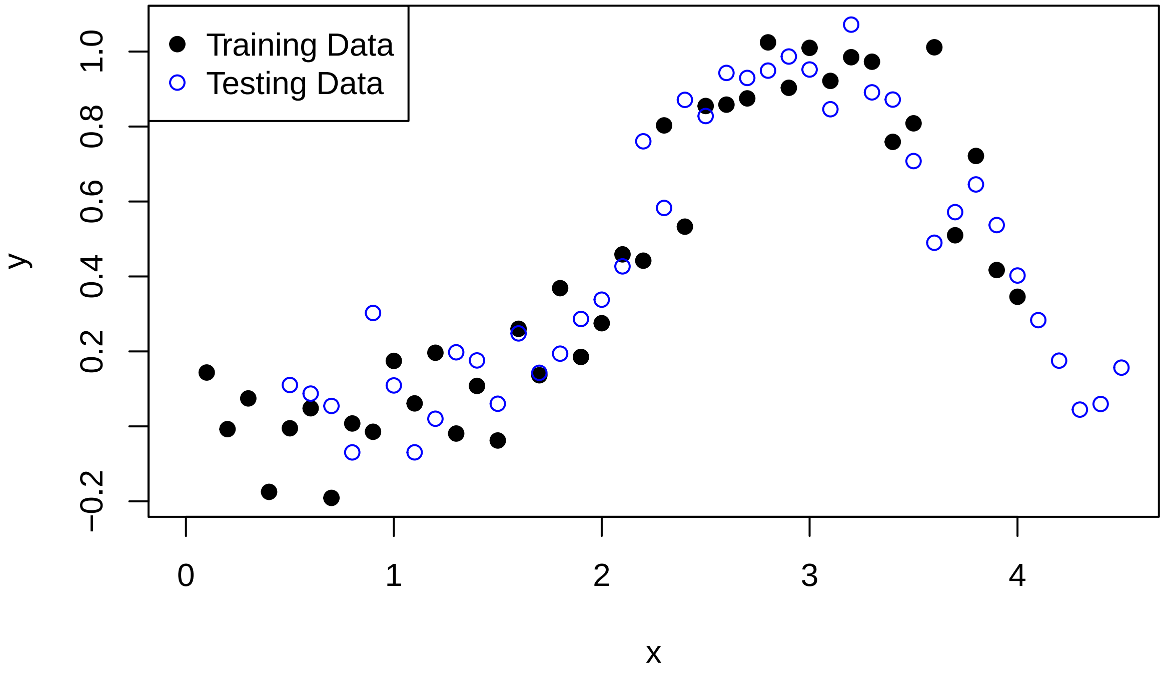


```r
n = length(x)
MSE = sum((y-fit1$fitted.values)^2)/n
MSE[2] = sum((y-fit2$fitted.values)^2)/n
MSE[3] = sum((y-fit3$fitted.values)^2)/n
MSE[4] = sum((y-fit4$fitted.values)^2)/n
MSE[5] = sum((y-fit5$fitted.values)^2)/n
MSE[6] = sum((y-fit6$fitted.values)^2)/n
K = c(1,2,3,5,10,20)
plot(K, MSE, type = "b", col=2,
     xlab="Model Complexity (Polynomial Order)")
```



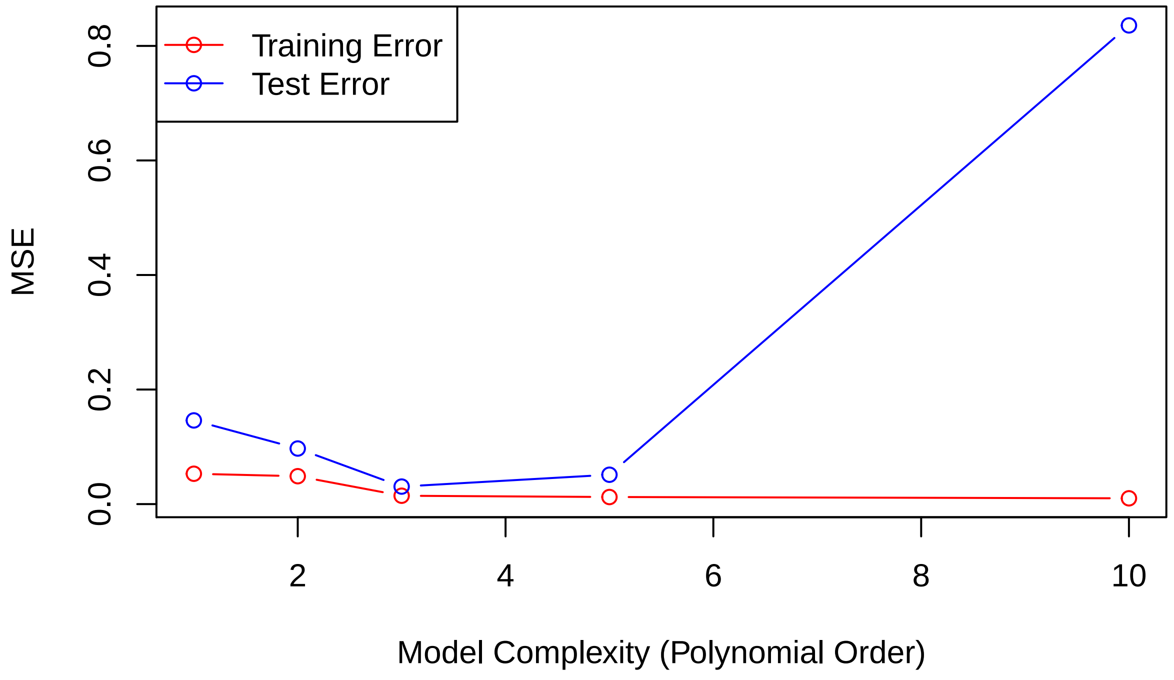
Generate new data and plot

```
xnew = seq(0.5,4.5,by=0.1)
set.seed(9999)
ynew = ffun(xnew) + 0.1*rnorm(length(xnew))
TestX = data.frame(x=xnew, y=ynew)
plot(x, y, pch=19, xlim=c(0,4.5),ylim=c(min(y,ynew),max(y,ynew)))
points(TestX$x, TestX$y, col=4)
legend("topleft", c("Training Data", "Testing Data"), pch=c(19,21), col=c(1,4))
```



Plot training and testing errors

```r
TestX$pred1 = predict(fit1, data.frame(x=xnew))
TestX$pred2 = predict(fit2, data.frame(x=xnew))
TestX$pred3 = predict(fit3, data.frame(x=xnew))
TestX$pred4 = predict(fit4, data.frame(x=xnew))
TestX$pred5 = predict(fit5, data.frame(x=xnew))
nnew = dim(TestX)[1]
TestMSE = sum((TestX$y-TestX$pred1)^2)/nnew
TestMSE[2] = sum((TestX$y-TestX$pred2)^2)/nnew
TestMSE[3] = sum((TestX$y-TestX$pred3)^2)/nnew
TestMSE[4] = sum((TestX$y-TestX$pred4)^2)/nnew
TestMSE[5] = sum((TestX$y-TestX$pred5)^2)/nnew
KK = c(1,2,3,5,10)
matplot(KK, cbind(MSE[1:5], TestMSE), type = "b", lty = 1, pch = 1,
        col=c(2,4), xlab="Model Complexity (Polynomial Order)", ylab="MSE")
legend("topleft", c("Training Error", "Test Error"), lty=1, pch=1, col=c(2,4))
```



Solution : Cross-Validation

A possible solution is to use cross-validation (CV). In its basic version, the so called *K-fold cross-validation*, the samples are randomly partitioned into K sets (called folds) of roughly equal size. A model is fit using all the samples except the first subset. Then, the prediction error of the fitted model is calculated using the first held-out samples. The same operation is repeated for each fold and the model's performance is calculated by averaging the errors across the different test sets. K is usually fixed at 5 or 10 . Cross-validation provides an estimate of the test error for each model. Cross-validation is one of the most widely-used method for model selection, and for choosing tuning parameter values.

The case where K=n corresponds to the so called *leave-one-out cross-validation (LOOCV)* method. In this case the test set contains a single observation.

The advantages of LOOCV are:

- It doesn't require random numbers to select the observations to test, meaning that it doesn't produce different results when applied repeatedly.
- It has far less bias than K-fold CV because it employs larger training sets containing n-1 observations each.

On the other side, LOOCV presents also some drawbacks:

- It is potentially quite intense computationally.
- Due to the fact that any two training sets share n-2 points, the models fit to those training sets tend to be strongly correlated with each other.

The code below gives a simple example for 10-fold cross-validation

```r
#install.packages("cvTools")
#run the above line if you don't have this library
library(cvTools)
```

```
## Loading required package: lattice

## Loading required package: robustbase
```

```r
#the number of folds
k = 10
folds = cvFolds(nrow(DataX), K=k)
DataX$holdoutpred = rep(0,nrow(DataX))

for(i in 1:k){

  #Set the validation set
  validation = DataX[folds$subsets[folds$which == i], ]

  #Set the training set
  train = DataX[folds$subsets[folds$which != i], ]

  #Get your new linear model (just fit on the train data)
  newlm = lm(y~x,data=train)
  #Get the predicitons for the validation set
  newpred = predict(newlm,newdata=validation)

  #Put the hold out prediction in the data set for later use
  DataX[folds$subsets[folds$which == i], ]$holdoutpred = newpred
}

head(DataX)
```

```
##     x            y holdoutpred
## 1 0.1  0.143642778 -0.12024454
## 2 0.2 -0.007335527 -0.05516462
## 3 0.3  0.074596051 -0.04865782
## 4 0.4 -0.174701244 -0.02266830
## 5 0.5 -0.005052069  0.01108787
## 6 0.6  0.048341664  0.03734802
```

Try to expand the above code into finding the optimal degree of freedom from 1 through 10

```
degree = seq(1,10,1)
rep_len = 50
k=10
TestError = matrix(0,length(degree),rep_len)

for (d in degree){
  test_error=NULL
  for(rep in 1:rep_len){
    DataX$holdoutpred = rep(0,nrow(DataX))
    folds = cvFolds(nrow(DataX), K=k)
    for(i in 1:k){
      validation = DataX[folds$subsets[folds$which == i], ]
      train = DataX[folds$subsets[folds$which != i], ]
      new_model = lm(y~poly(x,d), data=train)
      new_pred = predict(new_model, newdata=validation)
      DataX[folds$subsets[folds$which == i], ]$holdoutpred = new_pred
    }
    test_error = cbind(test_error, sum((DataX$holdoutpred - DataX$y)^2)/nrow(DataX))
  }
  TestError[d,] = test_error
}
```

Find the test error using Leave-one-out Cross-vlidation

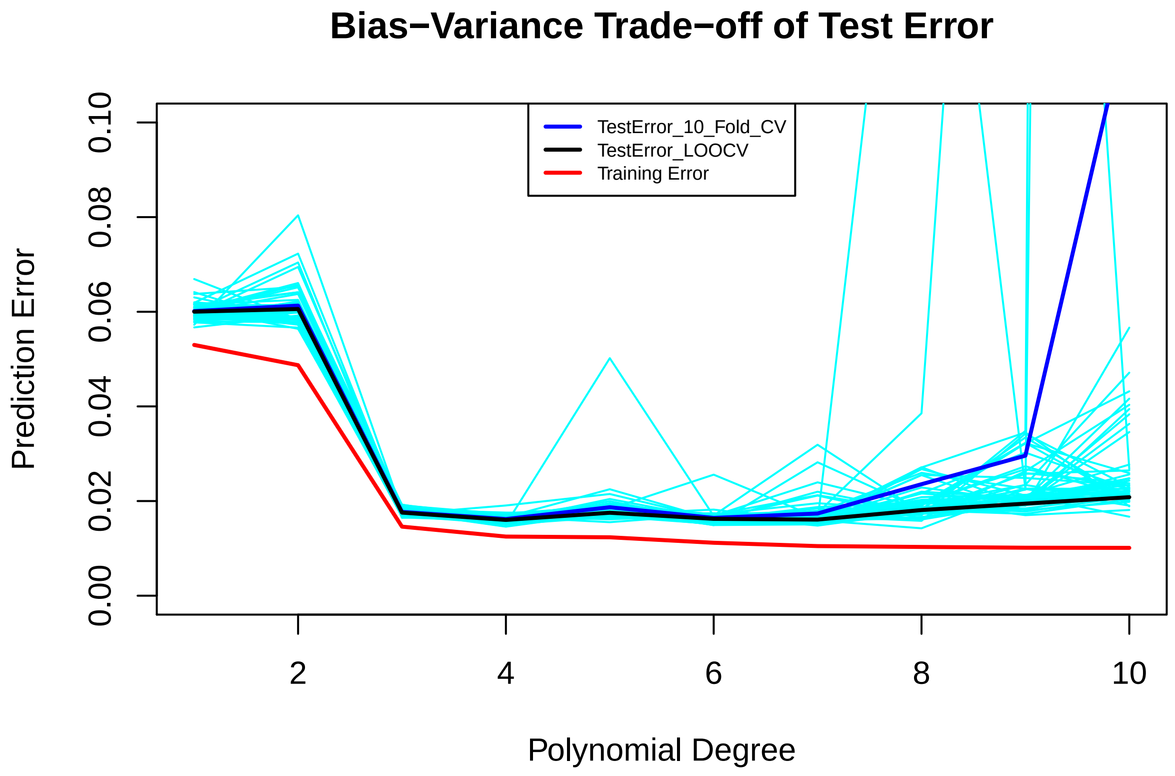
```
k=40
TestError_LOOCV = NULL

for (d in degree){
  DataX$holdoutpred = rep(0,nrow(DataX))
  for(i in 1:k){
    validation = DataX[i, ]
    train = DataX[-i, ]
    new_model = lm(y~poly(x,d), data=train)
    new_pred = predict(new_model, newdata=validation)
    DataX[i, ]$holdoutpred = new_pred
  }
  TestError_LOOCV = cbind(TestError_LOOCV, sum((DataX$holdoutpred-DataX$y)^2)/nrow(DataX))
}
```

Find the training error

```
TrainError = NULL
for(d in degree){
  fit = lm(y~poly(x,d), data = DataX)
  TrainError = c(TrainError, sum((DataX$y - fit$fitted.values)^2)/nrow(DataX))
}
```

Plot the training error and testing error regarding different cross-validation methods

```
matplot(1:10, TestError, main="Bias-Variance Trade-off of Test Error",
        type="l", lty=1, col=5, ylim = c(0, 0.1),
        xlab = "Polynomial Degree", ylab="Prediction Error")
lines(rowMeans(TestError), col=4, lwd=2)
lines(1:10, TestError_LOOCV, col=1, lwd=2)
lines(1:10, TrainError, col=2, lwd=2)
legend("top", lty=1, lwd=2, cex=0.6,
       c("TestError_10_Fold_CV", "TestError_LOOCV", "Training Error"),
       col=c(4, 1, 2))
```


Bias−Variance Trade−off of Test Error
TestError_10_Fold_CV
TestError_LOOCV
Training Error
Prediction Error
Polynomial Degree