

Exercises Statistical Methods for Bioinformatics


Part II, session I

March 24, 2020


Mini Theory Review:

(question 2.4.1)

For each of parts (a) through (d) indicate whether we would expect the performance of a flexible statistical learning method to be better or worse than a less flexible method (such as simple linear regression). Explain your answer:

1. The sample size n is very large, the number of parameters p is small. 
2. The number of predictors p is very large, and the number of observations is small.
3. The relationship between the predictors and the response is strongly non-linear.
4. The variance of the error terms, i.e. $\sigma^2 = \text{Var}(\epsilon)$, is very large.

(question 5.3) We now review the k -fold cross-validation.

1. Explain how k -fold cross-validation is implemented. 
2. What are the advantages and disadvantages when compared to the validation set approach; and the Leave-one-out-cross-validation (LOOCV)?



Practicals

(Partly from the labs of Chapter 5) The code is also provided. Load some relevant packages:

```
library("boot")
library("ISLR")
```

Maybe, you need to rst install the relevant packages in R, by typing:

```
install.packages("ISLR")
install.packages("boot")
```

Now start off with using a training and test set. The built-in `sample()`

function can be used to make a training set. You can make the random sampling process repeatable by setting the Random number generator's seed. In this case the dataset has 392 observations and we divide the dataset in 2 equal halves.

```
set.seed(1)
train=sample(392,196)
train
```

When loading the ISLR package you also loaded the Auto dataset. This is how you learn a model on the training set:

```
lm.fit=lm(mpg~horsepower,data=Auto,subset=train)
```

Now you can use the opposite of the training set (the test set), and the predict function to make predictions using the learned model.

```
test=Auto[-train,]
predictions=predict(lm.fit,test)
```

Now we will calculate a performance measure: the mean squared error (MSE):

$$MSE = \frac{1}{n} \sum_i^n (y_i - \hat{y}_i)^2$$

```
mean((test$mpg-predictions)^2)
```

- Rerun this code with different seeds. Does the performance measure vary importantly?

Let's now make a set of predictors with larger degrees of polynomials and see if these more complex models better t the data, both on the test and on the training set. The `poly()` function is a convenient tool for this in R. If you specify `y~poly(x,1)` in a formula, it stands for $y = \beta_0 + \beta_1 x + \epsilon$ as the model; `y~poly(x,2)` stands for $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \epsilon$ etc.

```
testMSE=rep(0,6)
trainMSE=rep(0,6)
for(i in 1:6){
  lm.fit=lm(mpg~poly(horsepower,i),data=Auto,subset=train)
  testMSE[i]=mean((test$mpg-predict(lm.fit,test))^2)
  trainMSE[i]=mean((Auto$mpg[train]-predict(lm.fit))^2)
}
```

You can plot the results in a straightforward way:

```
plot(testMSE,type="b",ylim=c(min(testMSE,trainMSE),max(testMSE,trainMSE)))
lines(trainMSE, type="b",col=2)
```

- Why do you think the test and train MSEs diverge?

- What do you think is the ideal complexity of the model? Is a quadratic or cubic term warranted?

Now we can do a simple cross-validation to check our results. By switching to the `glm()` function, which is normally used for generalized linear models, we can use the `cv.glm()` function, that implements cross-validation for us. When using the `glm` function, you can specify different error families to change the behaviour of the function, such as `family=binomial` for logistic regression. If we don't specify anything, the function behaves as `lm()`. The *LOOCV* below is a bit slow due to the many learning cycles.

```
loocv.error=rep(0,6)
cv10.error=rep(0,6)
for (i in 1:6){
  glm.fit=glm(mpg~poly(horsepower,i),data=Auto)
  loocv.error[i]=cv.glm(Auto,glm.fit)$delta[1]
  cv10.error[i]=cv.glm(Auto,glm.fit,K=10)$delta[1]
}
```

- Does the result agree with the training set approach?
- Any systematic difference between LOOCV and ten-fold CV? Is this what you would expect?

Bootstrap

Now we will use the bootstrap to test if the confidence interval of a statistic coincides with the value coming from a linear regression assuming Gaussian errors. For this we will use a simulated dataset. In this way we know exactly what we put in, and hence what we should get out. In this case we will violate one of the assumptions underlying the normal linear regression set-up: the normal distribution of errors. We will use a built-in function of a different distribution, the t-distribution, to create thicker tails, i.e. more values further away from the mean than expected under the normal distribution. Let's have a look at those distributions:

```
x <- seq(-4, 4, length=100)
hx <- dt(x,1.1)
hx2 <- dnorm(x,sd=1)
plot(hx2~x,type="l")
lines(hx~x,col=2)
```

First we generate the dataset:

```
set.seed(3)#make sure we get the same results
x = rnorm(1000)#some values, normal distribution around 0,sd=1
y = 1 + x + rt(1000,1.1) #simple relation, added noise is broad tailed! plot(x,y)
#broad tails give outliers
model = lm(y~x)
```

Now have a look at the estimated standard errors for the β_1 coefficient: `summary(model)`

- Is the association between x and y significant? How would you normally interpret this result?

In R we can find bootstrap functionality through the boot library with the boot function. To use the function we need to define a function to calculate the statistic of interest:

```
bootfun = function(data,b,formula){
  # b are the bootstrap sample indices
  d = data[b,] return(lm(d[,1]~d[,2], data = d)$coef[2]) # thats the beta1 coefficient
}
```

Now you can bootstrap using the boot function:

```
result=boot(data=data.frame(y,x), statistic=bootfun, R=1000)
#R is number of bootstrap samples
result
plot(result)
```

You can observe the confidence intervals given by the bootstrap approximation and the linear model:

```
boot.ci(result, index=1, type=c("bca"))#estimated confidence interval for bootstrap
confint(model)#confidence interval for linear model
```

- Do the confidence intervals match? Which value do you trust more and why?
- Change the artificial data above so that the error term has a normal distribution (`rnorm(1000,sd=1)` for example), and test if you get a different result.