Introduction to Statistical Learning 7/3/2017

Chapter 5: Resampling Methods

Name: Kamaru-Deen Lawal

Email: kamaru.ade.lawal@gmail.com

3

4

```
x = -2:2
y = c(1, 1, 2, 2, 1)
plot(x,y)
```

1 Let $f(\alpha) = \text{Var}(\alpha X + (1-\alpha)Y)$

$$\text{Var}(\sum_{i=1}^{n} \alpha_i X_i) = \sum_{i=1}^{n} \alpha_i^2 \,\text{Var}(X_i) + 2 \sum_{1 \le i \le j \le n} \alpha_i \alpha_j \,\text{Cov}(X_i, X_j) \Rightarrow f(\alpha) = \alpha^2\sigma_X^2 + (1-\alpha)^2\sigma_Y^2 + 2(\alpha)(1-\alpha)\sigma_{XY}$$

$$\begin{aligned} f'(\alpha) &= 2\alpha\sigma_X^2 - 2(1-\alpha)\sigma_Y^2 + 2(1-2\alpha)\sigma_{XY} = 0 \\ &\Rightarrow 2\alpha\sigma_X^2 - 2\sigma_Y^2 + 2\alpha\sigma_Y^2 + 2\sigma_{XY} - 4\alpha\sigma_{XY} = 0 \\ &\Rightarrow 2\alpha(\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}) = 2\sigma_Y^2 - 2\sigma_{XY} \\ &\Rightarrow \alpha = \frac{\sigma_Y^2 - \sigma_{XY}^2}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}} \end{aligned}$$

2

(a) $\frac{n-1}{n} = 1 - \frac{1}{n}$

(b) $\frac{n-1}{n} = 1 - \frac{1}{n}$

(c) $(1 - \frac{1}{n}) \cdot (1 - \frac{1}{n}) \cdots (1 - \frac{1}{n}) = (1 - \frac{1}{n})^n$

(d) $1 - (1 - \frac{1}{5})^5$

(e) $1 - (1 - \frac{1}{100})^{100}$

(f) $1 - (1 - \frac{1}{10000})^{10000}$

(g)

```
x = 1:100000
trans = function(x) {
  result = 1:100000
  for (i in 1:100000) {
    result[i] = (1-(1/i))^i
  }
  return(result)
}
y = trans(x)
plot(x,y)
```

$(1 - \frac{1}{i})^i$ levels off very quickly.

(h) There seems to be a fairly high probability that the fourth observation is contained in the bootstrap sample.

3

(a) Randomly divide the n observations into k groups of approximately equal size. The first fold is the validation set, and the other $k - 1$ sets are used to train the model. The error equals the average of the MSE's for all k validation sets.

(b) (i) Because we are only splitting the data into two sets the error is completely dependent on how we partition data. k-fold cross validation gets around this by partitioning the data in multiple ways and averaging the MSEs.

(ii) If n is large k-fold cross validation for say $n = 5$ or $n = 10$ is good because it requires less computation than fitting the model n times. k-fold cross validation also gives more accurate estimates of the test error. This is a direct result of the bias-variance trade off. In LOOCV the results will tend to be less biased because we are using all except one point to train the model (in a way we are overfitting, which yields a decrease in bias). By decreasing n we are increasing bias, and decreasing variance (test error).

4

Randomly select n points from the set of data points containing the (X, Y) points. Use the resulting data set to produce an estimate for the response, Y. Call this first data set Z_1^*. Repeat this process B times for some large value of B, in order to produce B different bootstrap data sets and B corresponding predictor estimates. We can then estimate the standard error of the predictor.

5

(a)

```
set.seed(1)
attach(Default)
glm.fit = glm(default ~ income + balance, family = binomial)
```

(b)

```
dim(Default)
train = sample(10000, 5000)
glm.fit2 = glm(default ~ income + balance, family = binomial, subset = train)
glm.pred = rep("No", 5000)
glm.probs = predict(glm.fit2, Default[-train,], type = "response")
glm.pred[glm.probs > .5] = "Yes"
default.test = default[-train]
mean(glm.pred != default.test) # .0286
```

(c)

```
logModel = function(seed) {
   set.seed(seed)
   train = sample(10000, 5000)
   glm.fit2 = glm(default ~ income + balance, family = binomial, subset = train)
   glm.pred = rep("No", 5000)
   glm.probs = predict(glm.fit2, Default[-train,], type = "response")
   glm.pred[glm.probs > .5] = "Yes"
```

```
   default.test = default[-train]
   print(mean(glm.pred != default.test))
}

logModel(2) # .0276
logModel(100) # .0258
logModel(57) # .027
```

All of the results seem to be very close to one another.

(d)

```
set.seed(777)
student01 = rep(0, 10000)
student01[student == "Yes"] = 1
train = sample(10000, 5000)
glm.fit3 = glm(default ~ income + balance + student01, family = binomial, subset = train)
glm.pred = rep("No", 5000)
glm.probs = predict(glm.fit3, Default[-train,], type = "response")
glm.pred[glm.probs > .5] = "Yes"
default.test = default[-train]
mean(glm.pred != default.test) # .0242
```

Adding a dummy variable for student did slightly decrease the test error.

6

(a)

```
set.seed(5)
train = sample(10000, 5000)
glm.fit = glm(default ~ income + balance, family = binomial, subset = train)
summary(glm.fit)
# income std. error: 7.146e-06
# balance std. error: 3.101e-04
```

(b)

```
boot.fn = function(data,index)
   return(coef(glm(default ~ income + balance, family = binomial, subset = index)))
```

(c)

```
library(boot)
?boot
boot(Default, boot.fn, 1000)
```

(d) The standard errors that I got using the bootstrap method were slightly lower than the values that i got using the summary function.

7

(a)

```
Weekly = read.csv("Weekly.csv", header = T, na.strings = "?")
attach(Weekly)
glm.fit = glm(Direction ~ Lag1 + Lag2, family = binomial, data = Weekly)
```

(b)

```
glm.fit2 = glm(Direction ~ Lag1 + Lag2, family = binomial, data = Weekly, subset =
    2:1089)
```

(c)

```
glm.pred = rep("No", 1)
train = 2:1089
glm.probs = predict(glm.fit2, Weekly[-train,], type = "response")
contrast(Direction)
glm.pred[glm.probs > .5] = "Up"
```

The point ended up being classified as "Up" when it is in fact "Down". so it was incorrectly classified.

(d)

```
fill in later
```

(e) fill in later

8

(a)

```
set.seed(1)
y = rnorm(100)
x = rnorm(100)
y = x - 2*x^2 + rnorm(100)
```

$n = 100,\ p = 1,\ y = x - 2x^2 + \epsilon$

(b)

```
plot(x,y)
```

The plot looks like a negative quadratic.

(c)

```
Data = data.frame(x,y)
set.seed(2)
# i
lm.fit = glm(y ~ x)
cv.glm(Data, lm.fit)$delta # 5.890979 5.888812
# ii
lm.fit = glm(y ~ x + I(x^2))
cv.glm(Data, lm.fit)$delta # 1.086596 1.086326
# iii
lm.fit = glm(y ~ x + I(x^2) + I(x^3))
cv.glm(Data, lm.fit)$delta # 1.102585 1.102227
# iv
lm.fit = glm(y ~ x + I(x^2) + I(x^3) + I(x^4))
cv.glm(Data, lm.fit)$delta # 1.114772 1.114334
```

(d)

```
set.seed(7)
#  i
lm.fit = glm(y ~ x)
cv.glm(Data, lm.fit)$delta # 5.890979 5.888812
```

```r
# ii
lm.fit = glm(y ~ x + I(x^2))
cv.glm(Data, lm.fit)$delta # 1.086596 1.086326
# iii
lm.fit = glm(y ~ x + I(x^2) + I(x^3))
cv.glm(Data, lm.fit)$delta # 1.102585 1.102227
# iv
lm.fit = glm(y ~ x + I(x^2) + I(x^3) + I(x^4))
cv.glm(Data, lm.fit)$delta # 1.114772 1.114334
```

The results are the same as what in part (c). This is because there is nothing random about the process since each time we fit the model we isolate one point, and all of the other points are part of the training.

(e) The second model had the smallest LOOCV error. This is what I expected since the real model is actually quadratic.

(f) The coefficient estimates of the linear and quadratic terms seem to be statistically significant. These results do agree with the results drawn from the CV data.