Lab 7

Burhan Ahmed Hanif

11:59PM March 31, 2019

```
rm(list=ls())
```

Generate $\mathbb{D}$ with $n = 100$ and $p = 1$ where x is created from iid realizations from a standard uniform, y comes from $f(x) = 3 - 4x$ and δ are iid realizations from a T distribution with 10 degrees of freedom.

```
set.seed(1997)
n = 100
p = 1
X = matrix(runif(n) , ncol = 1)
f_x = 3 - 4 * X
y = f_x + rt(n, df = 10)
```

Run the linear model using `lm` and compute b, RMSE and R^2.

```
linear_mod = lm(y ~ X)
coef(linear_mod)
```

```
## (Intercept)           X
##    2.855769   -3.855183
```

```
summary(linear_mod)$sigma
```

```
## [1] 1.308767
```

```
summary(linear_mod)$r.squared
```

```
## [1] 0.4044066
```

Progressively add columns of x (as draws from a standard uniform), run the linear model, and show R^2 goes to 1 and s_e goes to zero. Save the s_e in a vector called `in_sample_s_e`.

```
in_sample_s_e = array(NA, n - 2)
linear_mods = list()
for (j in 1 : (n - 2)){
  X = cbind(X, runif(n))
  linear_mods[[j]] = lm(y ~ ., data.frame(X))
  in_sample_s_e[j] = sd(linear_mods[[j]]$residuals)
}
dim(X)
```

```
## [1] 100  99
```

```
summary(linear_mods[[j]])$r.squared
```

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

```
in_sample_s_e
```

```
##  [1] 1.28649282 1.28232218 1.27201474 1.25707441 1.23002701 1.22829460
##  [7] 1.22670007 1.22643605 1.21086228 1.16906116 1.16899113 1.14450895
## [13] 1.14430182 1.14429580 1.12528678 1.12513648 1.11418967 1.11103788
## [19] 1.10707910 1.10617314 1.10593501 1.10383989 1.09059722 1.05983015
## [25] 1.05668551 1.04443529 1.03545946 1.03542747 1.01766877 1.00972008
```

```
## [31] 0.94628966 0.94173458 0.93946800 0.92946819 0.91433407 0.91400464
## [37] 0.91178872 0.90295678 0.90275680 0.88591059 0.87734241 0.87731196
## [43] 0.87038874 0.87012819 0.86867611 0.85599859 0.84341701 0.84331509
## [49] 0.84231901 0.83898042 0.83429865 0.83191842 0.79093216 0.78712763
## [55] 0.76614183 0.74416204 0.74347964 0.74237630 0.74050592 0.72910998
## [61] 0.72569710 0.71281965 0.71163428 0.70538923 0.70016453 0.58618111
## [67] 0.58106388 0.56639234 0.55400937 0.55397187 0.54946747 0.54303693
## [73] 0.54086858 0.53230397 0.52439764 0.52245609 0.52184794 0.51818504
## [79] 0.51403634 0.51239241 0.48767310 0.46032595 0.44978526 0.32203624
## [85] 0.31974154 0.31231435 0.31111980 0.30818469 0.30506651 0.27027212
## [91] 0.22558040 0.21105584 0.19626704 0.18477877 0.17373682 0.14093467
## [97] 0.03613263 0.00000000
```

```r
d = diff(in_sample_s_e)
all(d < 0)
```

```
## [1] TRUE
```

Compute a corresponding vector `oos_s_e` and show that it is increasing (for the most part) in degrees of freedom.

```r
n_star = 1e5
p = 1
X_star = matrix(runif(n_star) , ncol = 1)
f_x_star = 3 - 4 * X_star
y_star = f_x_star + rt(n_star, df = 10)
oos_s_e = array(NA, n - 2)
for (j in 1 : (n - 2)){
  X_star = cbind(X_star, runif(n_star))
  y_hat_star = predict(linear_mods[[j]], data.frame(X_star))
  oos_s_e[j] = sd(y_star - y_hat_star)
}
oos_s_e
```

```
##  [1]  1.138872  1.145037  1.158311  1.174007  1.203102  1.205684  1.209192
##  [8]  1.211103  1.245249  1.301276  1.300797  1.306607  1.306549  1.306628
## [15]  1.333529  1.332744  1.363185  1.365351  1.371233  1.370490  1.372865
## [22]  1.375974  1.379607  1.408161  1.394222  1.426631  1.402238  1.399934
## [29]  1.438706  1.435763  1.546897  1.543536  1.543032  1.508717  1.537036
## [36]  1.538435  1.543836  1.536747  1.536909  1.576790  1.566087  1.565106
## [43]  1.593066  1.593993  1.599078  1.626604  1.616842  1.617316  1.625600
## [50]  1.618205  1.614296  1.593533  1.691716  1.717884  1.821657  1.796961
## [57]  1.787966  1.817988  1.836788  1.852059  1.863737  1.838741  1.846605
## [64]  1.912911  1.931040  2.144888  2.156903  2.236306  2.314845  2.314554
## [71]  2.324415  2.364255  2.376579  2.488367  2.417481  2.491206  2.500137
## [78]  2.420275  2.381528  2.395049  2.737752  2.694287  2.893602  3.232287
## [85]  3.276438  3.303585  3.417506  3.397106  3.440856  3.663598  3.941270
## [92]  4.244160  4.170434  4.293145  4.653674  5.214444 10.300784 14.797485
```

```r
d = diff(oos_s_e)
all(d > 0)
```

```
## [1] FALSE
```

Validate the linear model for the Boston housing data.

```r
Xy = MASS::Boston
K = 10
```

```
test_indices = sample(1 : nrow(Xy), 1 / K * nrow(Xy))
train_indices = setdiff(1 : nrow(Xy), test_indices)
Xy_train = Xy[train_indices, ]
Xy_test = Xy[test_indices, ]
lin_mod = lm(medv ~ ., Xy_train)
lin_mod
```

```
##
## Call:
## lm(formula = medv ~ ., data = Xy_train)
##
## Coefficients:
## (Intercept)         crim           zn        indus         chas
##   37.904367    -0.088225     0.038125     0.018173     2.642703
##         nox           rm          age          dis          rad
##  -17.686963     3.888130    -0.003146    -1.448199     0.316541
##         tax      ptratio        black        lstat
##   -0.013130    -1.007799     0.008375    -0.540627
```

```
sd(lin_mod$residuals)
```

```
## [1] 4.644024
```

```
y_hat_test = predict(lin_mod, Xy_test)
sd(Xy_test$medv - y_hat_test)
```

```
## [1] 5.155728
```

```
dim(Xy)
```

```
## [1] 506  14
```

Let x be iid realizations from a $U(0,5)$, y comes from $f(x) = 3 - 4x + 2x^2$ and ϵ are iid realizations from a standard normal distribution. With no limit on the number of samples you cant take, use regular OLS *without a quadratic term*, find the true $h^*(x)$ (there will be no sampling variability at $n \to \infty$ and find the oos variance of the residuals.

```
n = 1e6
X = cbind(rep(1,n), runif(n,0,5))
x_1= X[ ,2]
f_x = 3 - 4*x_1 + 2*(x_1^2)
epsilon = rnorm(n)
y = f_x + epsilon
b = solve(t(X) %*% X) %*% t(X) %*% y
h_star_x = b[1,1] + b[2,1]*x_1
y_hat = b %*% x_1
n_star = 1e6
p=1
X_star = matrix(runif(n_star , 0 , 5), ncol = 1)
f_x_star = 3- 4*X_star + 2*(X_star^2)
y_star = f_x_star + rnorm(n_star)
y_hat_star = predict(lm(y_star ~ ., data.frame(x_1)) , data.frame(X_star))
oos_s_e = sd(y-y_hat_star)
oos_s_e
```

```
## [1] 9.487416
```

Was there any overfitting in the previous exercise? #yes there was overfitting because the RMSE approches 0 means no error but compared to the oose the opposite is happening the error is getting largee.

Find the error due to misspecification and due to ignorance expressed as variance of components of the residuals.

```
error_Mspec_igor = sd(y - f_x)
error_Mspec_igor
```

```
## [1] 1.000092
```

At $n = 100$, find the error due to estimation, due to misspecification and due to ignorance expressed as variance of components of the residuals.

```
n = 100
X = cbind(rep(1,n), runif(n,0,5))
x_1= X[ ,2]
f_x = 3 - 4*x_1 + 2*(x_1^2)
epsilon = rnorm(n)
y = f_x + epsilon
b = solve(t(X) %*% X) %*% t(X) %*% y
b
```

```
##           [,1]
## [1,] -5.493995
## [2,]  5.976314
```

```
h_star_x = b[1,1] + b[2,1]*x_1
error_estima = sd(y - h_star_x)
error_estima
```

```
## [1] 3.819132
```

Do the variances add up to the total variance of the residual?

```
tot_error = sd(y - y_hat)
tot_error
```

```
## [1] 18.73321
```

Validate the linear model for the Boston housing data where each feature is also modeled with a squared feature.

```
X = MASS::Boston
y = X$medv
X$medv = NULL
X = cbind(X, X^2)
colnames(X)[14 : 26] = paste(colnames(X)[1 : 13], "_sq", sep = "")
X$chas_sq = NULL
K = 10
test_indices = sample(1 : nrow(Xy), 1 / K * nrow(Xy))
train_indices = setdiff(1 : nrow(Xy), test_indices)
X_train = X[train_indices, ]
y_train = y[train_indices]
X_test = X[test_indices, ]
y_test = y[test_indices]
lin_mod = lm(y_train ~ ., X_train)
#lin_mod
paste("Residuals with squared model", round(sd(lin_mod$residuals),5))
```

```
## [1] "Residuals with squared model 3.82843"
```

```
y_hat_test = predict(lin_mod, X_test)
paste("deviation after running the squared model " , round(sd(y_test - y_hat_test),5))
```

```
## [1] "deviation after running the squared model  3.38692"
```

Validate the linear model for the Boston housing data where each feature is also modeled with a squared feature and a cubed feature.

```
X = MASS::Boston
y = X$medv
X$medv = NULL
X = cbind(X, X^2 , X^3)
colnames(X)[14 : 26] = paste(colnames(X)[1 : 13], "_sq", sep = "")
colnames(X)[27 : 39] = paste(colnames(X)[1 : 13], "_cube", sep = "")
X$chas = NULL
X$chas_sq = NULL
X$chas_cube = NULL

K = 10
test_indices = sample(1 : nrow(X), 1 / K * nrow(X))
train_indices = setdiff(1 : nrow(X), test_indices)
X_train = X[train_indices, ]
y_train = y[train_indices]
X_test = X[test_indices, ]
y_test = y[test_indices]
lin_mod = lm(y_train ~ ., X_train)
#lin_mod
paste("Residuals with cube model", round(sd(lin_mod$residuals),5))
```

```
## [1] "Residuals with cube model 3.70767"
```

```
y_hat_test = predict(lin_mod, X_test)
paste("deviation after running the cube model " , round(sd(y_test - y_hat_test),5))
```

```
## [1] "deviation after running the cube model  3.53741"
```

Validate the linear model for the Boston housing data where each feature is also modeled with a squared feature and a cubed feature and a log(x + 1) feature and an exponential feature.

```
X = MASS::Boston
y = X$medv
X$medv = NULL
X$chas = NULL
X = cbind(X, X^2, X^3, log1p(X))
colnames(X)[13 : 24] = paste(colnames(X)[1 : 12], "_sq", sep = "")
colnames(X)[25 : 36] = paste(colnames(X)[1 : 12], "_cube", sep = "")
colnames(X)[37 : 48] = paste(colnames(X)[1 : 12], "_log", sep = "")
K = 10
test_indices = sample(1 : nrow(X), 1 / K * nrow(X))
train_indices = setdiff(1 : nrow(X), test_indices)
X_train = X[train_indices, ]
y_train = y[train_indices]
X_test = X[test_indices, ]
y_test = y[test_indices]
lin_mod = lm(y_train ~ ., X_train)
paste("Residuals of log model", round(sd(lin_mod$residuals), 5))
```

```
## [1] "Residuals of log model 3.44629"
```

```r
y_hat_test = predict(lin_mod, X_test)
paste("Deviation after running log model", round(sd(y_test - y_hat_test), 5))
```

```
## [1] "Deviation after running log model 4.67382"
```

Why do we need to log $x + 1$? Why not use log(x)? #if we have a point 0 then log(0) would diverge to negative infinity but log(x+1) shouldnt really have problem but we would have to account for the +1