Lab 7

Your Name Here

11:59PM March 31, 2019

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.

```
n = 100
X = matrix(runif(n), ncol = 1, nrow = 100)
f_X = 3 - 4 * X
y = f_X + rt(n, df = 10)

y
```

```
##               [,1]
##   [1,]  1.89486467
##   [2,]  2.67931854
##   [3,]  3.14268010
##   [4,] -1.45924276
##   [5,]  2.31460495
##   [6,]  0.68047412
##   [7,]  0.89420942
##   [8,]  1.52218934
##   [9,] -0.17080856
##  [10,] -1.91472219
##  [11,]  2.58668659
##  [12,]  0.24466702
##  [13,] -0.99567654
##  [14,]  0.54493895
##  [15,]  1.84777956
##  [16,] -2.07187631
##  [17,]  3.69283948
##  [18,]  0.29507399
##  [19,]  0.84734168
##  [20,] -0.17234055
##  [21,]  1.68240454
##  [22,]  1.77908402
##  [23,]  0.17844456
##  [24,]  3.08688116
##  [25,]  0.54666637
##  [26,]  2.62581377
##  [27,]  2.94699366
##  [28,]  1.88108620
##  [29,]  0.93911777
##  [30,]  1.06467204
##  [31,]  2.37920299
##  [32,]  2.57545003
##  [33,]  0.82447662
##  [34,]  0.80380153
##  [35,]  4.31214018
##  [36,] -0.19852089
```

```
## [37,] -0.76266167
## [38,]  3.79688040
## [39,]  2.37185476
## [40,]  0.42814441
## [41,] -1.57220092
## [42,]  0.89543302
## [43,]  2.37717668
## [44,]  1.54691660
## [45,] -0.61014206
## [46,] -2.90674909
## [47,] -2.33770680
## [48,] -2.73496828
## [49,] -0.91059955
## [50,]  0.23536510
## [51,]  3.91704629
## [52,]  0.46795599
## [53,]  1.55807912
## [54,]  1.22993018
## [55,]  3.29834945
## [56,]  1.53763342
## [57,] -2.19495027
## [58,]  0.94945208
## [59,]  0.70038534
## [60,]  0.01269109
## [61,] -0.36379312
## [62,]  0.09884249
## [63,] -0.37964385
## [64,]  0.64728791
## [65,]  3.51919486
## [66,]  1.31145004
## [67,]  0.95598513
## [68,]  2.35483833
## [69,] -0.93038618
## [70,]  3.71144196
## [71,]  0.12829067
## [72,]  1.59516918
## [73,]  0.32250588
## [74,] -0.40812217
## [75,] -1.97181729
## [76,]  2.25532578
## [77,]  1.06657831
## [78,] -1.97621541
## [79,]  2.82126782
## [80,]  3.04229741
## [81,] -0.71474034
## [82,]  0.07641098
## [83,]  2.22561451
## [84,]  5.05765102
## [85,]  0.07657568
## [86,]  0.18452663
## [87,]  0.99168758
## [88,] -2.43219545
## [89,]  0.10921618
## [90,]  3.50261347
```

```
##  [91,]  1.30788344
##  [92,]  3.22387267
##  [93,]  3.10695802
##  [94,] -0.10693205
##  [95,] -0.36458447
##  [96,]  0.39672841
##  [97,]  0.62998409
##  [98,]  1.63547279
##  [99,]  2.07276902
## [100,]  3.92229339
```

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

```r
model = lm(y ~ X)
coef(model)
```

```
## (Intercept)           X
##     3.30136    -4.59501
```

```r
summary(model)$sigma
```

```
## [1] 1.098557
```

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

```
## [1] 0.6030613
```

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`.

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

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

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

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

```r
s_e
```

```
##   [1] 1.08652454 1.08494164 1.08491236 1.06705438 1.04750193 1.04211466
##   [7] 1.04084270 1.03708051 1.03320162 1.03187525 1.03027187 1.02409504
##  [13] 0.98479050 0.98217274 0.98101119 0.98036302 0.97849995 0.97840883
##  [19] 0.97222408 0.97217208 0.96329941 0.95741673 0.94541934 0.92719644
##  [25] 0.92149788 0.91869811 0.91865063 0.88833694 0.86775788 0.86435994
##  [31] 0.83976957 0.83971127 0.83970126 0.78828542 0.78779443 0.78403584
##  [37] 0.76741967 0.76741690 0.76342700 0.76185020 0.76175052 0.72254693
##  [43] 0.71577096 0.71435614 0.70497435 0.70102497 0.70102349 0.70079152
##  [49] 0.69983358 0.69693790 0.69650239 0.69015648 0.68464156 0.63381755
##  [55] 0.63182660 0.61734241 0.61277705 0.58902695 0.58708058 0.58704824
##  [61] 0.58076806 0.57069286 0.56566840 0.55820170 0.55500249 0.55347035
##  [67] 0.54261311 0.54241978 0.53886085 0.53798889 0.53761831 0.53687652
##  [73] 0.52824707 0.52824537 0.49352795 0.49352794 0.46979977 0.46177604
##  [79] 0.46159521 0.45611321 0.40595876 0.39332944 0.39330345 0.39329583
##  [85] 0.38376323 0.32426958 0.28601949 0.27407801 0.27197155 0.24708526
##  [91] 0.24373839 0.20650194 0.20389188 0.20086875 0.17866565 0.07439792
##  [97] 0.05893142 0.00000000
```

```r
d = diff(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 = 100
X_star = matrix(runif(n_star), ncol = 1, nrow = 100)
f_X = 3 - 4 * X_star
y_star = f_X + rt(n_star, df = 10)

s_e = array(NA, n - 2)

#for (j in 1 : (n - 2)){
 # X_star = cbind(X_star, runif(n_star))
 # predict(lms)
#}
```

Validate the linear model for the Boston housing data.

```r
X_y = MASS::Boston
K = 10
test_indeces = sample(1:nrow(X_y), 1/K * nrow(X_y))
train_indeces = setdiff(1 : nrow(X_y), test_indeces)

xytrain = X_y[train_indeces, ]
xytest = X_y[test_indeces, ]

#sort(test_indeces)
#sort(train_indeces)

lm = lm(medv ~ ., xytrain)

summary(lm)$sigma
```

```
## [1] 4.616719
```

```
yhat_test = predict(lm, xytest)

sd(xytest$medv - yhat_test)
```

```
## [1] 5.943407
```

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 = 1e5
K = 10
x = runif(n,0,5)
x = cbind(1,x)
test_indices = sample(1 : nrow(x), 1 / K * nrow(X_y))
train_indices = setdiff(1 : nrow(x), test_indices)
X_test = x[test_indices, ]
X_train = x[train_indices, ]
f_x = 3-4*x[, 2]^2
e = rnorm(n)
y= f_x + e
b = solve(t(x) %*% x) %*% t(x) %*% y

h_star = b[1] + b[2]*x[, 2]
yhat = b[1] + b[2]*xytest
```

Was there any overfitting in the previous exercise?

No

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

```
mispecError1 = sd(f_x - h_star)
ignoranceError1 = sd(y - f_x)
```

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 = runif(n,0,5)
x = cbind(1,x)
test_indices = sample(1 : nrow(x), 1 / K * nrow(X_y))
train_indices = setdiff(1 : nrow(x), test_indices)
x_test = x[test_indices, ]
x_train = x[train_indices, ]
f_x = 3-4*x[, 2]^2
e = rnorm(n)
y= f_x + e
b = solve(t(x) %*% x) %*% t(x) %*% y
```

```
h_star = b[1] + b[2]*x[, 2]
yhat = b[1] + b[2]*x_test
mispecError2 = sd(f_x - h_star)
ignoranceError2 = sd(y - f_x)
```

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

```
model = lm(y~x)
yhat = predict(model, data = x)
var = sd(y - yhat)
sum((y-yhat)^2)
```

```
## [1] 5378.504
```

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

```
X = cbind(X, X^2)

X = MASS::Boston

y = X$medv

X$medv = NULL

K = 10
test_indeces = sample(1:nrow(X), 1/K * nrow(X))
train_indeces = setdiff(1 : nrow(X), test_indeces)

xtrain = X[train_indeces, ]
ytrain = y[train_indeces]
xtest = X[test_indeces, ]
ytest = y[train_indeces]



#sort(test_indeces)
#sort(train_indeces)

lm = lm(ytrain ~ ., xtrain)

summary(lm)$sigma
```

```
## [1] 4.85664
```

```
yhat_test = predict(lm, xtest)

sd(xytest$medv - yhat_test)
```

```
## [1] 9.384942
```

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)
colnames(X)[14 : 26] = paste(colnames(X)[1 : 13], "_sq", sep = "")
X$chas_sq = NULL
K = 10
test_indices = sample(1 : nrow(X_y), 1 / K * nrow(X_y))
train_indices = setdiff(1 : nrow(X_y), test_indices)
X_train = X[train_indices, ]
y_train = y[train_indices]
X_test = X[test_indices, ]
y_test = y[test_indices]
lin_mod = lm('y_train ~ .', X_train)
#lin_mod
sd(lin_mod$residuals)
```

```
## [1] 3.817911
```

```
y_hat_test = predict(lin_mod, X_test)
sd(y_test - y_hat_test)
```

```
## [1] 3.624386
```

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.

```
#TO-DO
```

Why do we need to log $x + 1$? Why not use log(x)?

#TO-DO