Lab 7

Adriana Sham

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.

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

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)
}

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

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

```
tail(in_sample_s_e)
```

```
## [1] 0.19626704 0.18477877 0.17373682 0.14093467 0.03613263 0.00000000
```

```
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.

```
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)
}

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

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

Validate the linear model for the Boston housing data.

```
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
##   35.679799    -0.105926     0.044428     0.036199     1.785549
##         nox           rm          age          dis          rad
##  -18.514293     4.075433    -0.000166    -1.454425     0.310478
##         tax      ptratio        black        lstat
##   -0.012749    -0.989053     0.009001    -0.492947
```

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

```
## [1] 4.725718
```

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

```
## [1] 4.344533
```

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)))
X1 = X[,2]
f_x = 3 - 4*X1 + 2*(X1^2)
epsilon = rnorm(n)
y = f_x + epsilon
b = solve(t(X) %*% X) %*% t(X) %*% y
h_star_x = b[1,1] + b[2,1]*X1
yhat = b %*% X1

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(X1)), data.frame(X_star))
oos_s_e = sd(y - y_hat_star)
oos_s_e
```

```
## [1] 9.485065
```

Was there any overfitting in the previous exercise?

Yes, it is overfitting since the RMSE aproaches 0 which means no error in the data set. With the out of sample RMSE, happens the opposite instead of going to 0, it gets larger and larger.

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

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

```
## [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)))
X1 = X[,2]
f_x = 3 - 4*X1 + 2*(X1^2)
epsilon = rnorm(n)
y = f_x + epsilon
b = solve(t(X) %*% X) %*% t(X) %*% y
b
```

```
##           [,1]
## [1,] -5.086237
## [2,]  5.821471
```

```
h_star_x = b[1,1] + b[2,1]*X1
e_estim = sd(y - h_star_x)
e_estim
```

```
## [1] 3.865573
```

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

```
tot_e = sd(y - yhat)
tot_e
```

```
## [1] 18.82731
```

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(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 from squared model:", round(sd(lin_mod$residuals), 5))
```

```
## [1] "Residuals from squared model: 3.882"
```

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

```
## [1] "Deviation after running squared model on data 2.98426"
```

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]
```

```r
lin_mod = lm(y_train ~ ., X_train)
paste("Residuals of cubed model from training data:", round(sd(lin_mod$residuals)), 5)
```

```
## [1] "Residuals of cubed model from training data: 4 5"
```

```r
y_hat_test = predict(lin_mod, X_test)
paste("Deviation after running cubed model on new data:", round(sd(y_test - y_hat_test)), 5)
```

```
## [1] "Deviation after running cubed model on new data: 4 5"
```

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.

```r
#TO-DO
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 from training data:", round(sd(lin_mod$residuals), 5))
```

```
## [1] "Residuals of log model from training data: 3.39027"
```

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

```
## [1] "Deviation after running log model on new data: 3.76739"
```

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

If one of the data points is zero log(x) will diverge to negative infinity. All of the feature data points are positive so x+1 poses no issues.