Predictive Practical Set 5

Diyasha Basu

26-02-2026

Problem to demonstrate the role of qualitative (ordinal) predictors in addition to quantitative predictors in multiple linear regression.

Consider “diamonds” data set in R. It is in the ggplot2 package. Make a list of all the ordinal categorical variables. Identify the response.

The ordinal categorical variables are **cut** (Fair < Good < Very Good < Premium < Ideal), **color** (D < E < F < G < H < I < J), and **clarity** (I1 < SI2 < SI1 < VS2 < VS1 < VVS2 < VVS1 < IF). The response variable is **price**.

(a) Linear regression of price on cut

```r
contrasts(diamonds$cut) = contr.sdif(5)
ord_mod = lm(price ~ cut, data = diamonds)
summary(ord_mod)

##
## Call:
## lm(formula = price ~ cut, data = diamonds)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##  -4258  -2741  -1494   1360  15348
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4062.24      25.40 159.923  < 2e-16 ***
## cut2-1       -429.89     113.85  -3.776  0.00016 ***
## cut3-2         52.90      67.10   0.788  0.43055
## cut4-3        602.50      49.39  12.198  < 2e-16 ***
## cut5-4      -1126.72      43.22 -26.067  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3964 on 53935 degrees of freedom
## Multiple R-squared:  0.01286,    Adjusted R-squared:  0.01279
## F-statistic: 175.7 on 4 and 53935 DF,  p-value: < 2.2e-16
```

Fitted model: The fitted model is:

$$\widehat{price} = 4062.24 - 429.89 \cdot C_{21} + 52.90 \cdot C_{32} + 602.50 \cdot C_{43} - 1126.72 \cdot C_{54}$$

where the coefficients represent successive differences between adjacent cut levels (Good-Fair, Very Good-Good, Premium-Very Good, Ideal-Premium).

(b) Test: Is expected price of Premium cut significantly different from Ideal cut?

```
lvl = as.numeric(diamonds$cut)
code_fun = function(x) {
  if(x-5==0) {
    return(c(0,0,0,0))
  } else if(x-5==-1) {
    return(c(1,0,0,0))
  } else if(x-5==-2) {
    return(c(1,1,0,0))
  } else if(x-5==-3) {
    return(c(1,1,1,0))
  } else {
    return(c(1,1,1,1))
  }
}
coded_prep = t(sapply(lvl, code_fun))
colnames(coded_prep) = c("Premium", "Very_Good", "Good", "Fair")
diamonds2 = cbind(diamonds, coded_prep)
ord_mod1 = lm(price ~ Premium + Very_Good + Good + Fair, data = diamonds2)
summary(ord_mod1)

##
## Call:
## lm(formula = price ~ Premium + Very_Good + Good + Fair, data = diamonds2)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##  -4258  -2741  -1494   1360  15348
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3457.54      27.00 128.051  < 2e-16 ***
## Premium      1126.72      43.22  26.067  < 2e-16 ***
## Very_Good    -602.50      49.39 -12.198  < 2e-16 ***
## Good          -52.90      67.10  -0.788  0.43055
## Fair          429.89     113.85   3.776  0.00016 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3964 on 53935 degrees of freedom
## Multiple R-squared:  0.01286,    Adjusted R-squared:  0.01279
## F-statistic: 175.7 on 4 and 53935 DF,  p-value: < 2.2e-16
```

The coefficient of `Premium` tests whether Premium cut differs from Ideal cut. The t-test p-value for `Premium` indicates it **is** significantly different from Ideal cut ($p < 0.05$).

(c) Expected price of a diamond of Ideal cut

```
ideal_price = coef(ord_mod1)["(Intercept)"]
cat("Expected price of Ideal cut diamond: $", round(ideal_price, 2))
```

```
## Expected price of Ideal cut diamond: $ 3457.54
```

(d) Add “table” as predictor

```r
ord_mod2 = lm(price ~ Premium + Very_Good + Good + Fair + table, data =
diamonds2)
summary(ord_mod2)
```

```
##
## Call:
## lm(formula = price ~ Premium + Very_Good + Good + Fair + table,
##     data = diamonds2)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##  -5630  -2694  -1458   1346  15690
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -6563.672    517.450 -12.685  < 2e-16 ***
## Premium       626.220     50.215  12.471  < 2e-16 ***
## Very_Good    -461.015     49.761  -9.265  < 2e-16 ***
## Good         -185.162     67.220  -2.755  0.00588 **
## Fair          365.568    113.504   3.221  0.00128 **
## table         179.105      9.236  19.393  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3950 on 53934 degrees of freedom
## Multiple R-squared:  0.0197, Adjusted R-squared:  0.01961
## F-statistic: 216.7 on 5 and 53934 DF,  p-value: < 2.2e-16
```

Fitted model:

$$\widehat{price} = \hat{\beta}_0 + \hat{\beta}_1 \cdot Premium + \hat{\beta}_2 \cdot VeryGood + \hat{\beta}_3 \cdot Good + \hat{\beta}_4 \cdot Fair + \hat{\beta}_5 \cdot table$$

Coefficients : $\hat{\beta}_0 = -6563.672$, $\hat{\beta}_1 = 626.220$, $\hat{\beta}_2 = -461.015$, $\hat{\beta}_3 = -185.162$, $\hat{\beta}_4 = 365.568$ and $\hat{\beta}_5 = 179.105$

(e) Test significance of “table”

From the summary above, the t-test for `table` shows its p-value. Since p < 0.05, then `table` is a significant predictor of diamond price.

(f) Average estimated price for average table value, Fair cut

```r
avg_table = mean(diamonds2$table)
cat("Average table value:", avg_table, "\n")
```

```
## Average table value: 57.45718
```

```r
# Fair cut: Premium=1, Very_Good=1, Good=1, Fair=1
new_data = data.frame(Premium=1, Very_Good=1, Good=1, Fair=1,
```

```
table=avg_table)
pred_price = predict(ord_mod2, newdata=new_data)
cat("Expected price (Fair cut, avg table): $", round(pred_price, 2))

## Expected price (Fair cut, avg table): $ 4072.8
```

Problem to demonstrate the utility of K nearest neighbour regression over least squares regression.

Setup: Generate data

```
set.seed(123)
n = 1000
x1_ = rnorm(n, 0, 2)
x2_ = rpois(n, 1.5)
epi_ = rnorm(n, 0, 1)
y_ = -2 + 1.4*x1_ - 2.6*x2_ + epi_

data1 = data.frame(y_, x1_, x2_)
train_data = data1[1:800, ]
test_data  = data1[801:1000, ]
```

1. Multiple Linear Regression

```
model_train = lm(y_ ~ x1_ + x2_, data = train_data)
summary(model_train)

##
## Call:
## lm(formula = y_ ~ x1_ + x2_, data = train_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0727 -0.6573 -0.0125  0.6921  3.2412
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2.07300    0.05382  -38.52   <2e-16 ***
## x1_          1.38207    0.01767   78.21   <2e-16 ***
## x2_         -2.55584    0.02768  -92.34   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.98 on 797 degrees of freedom
## Multiple R-squared:  0.9492, Adjusted R-squared:  0.9491
## F-statistic:  7445 on 2 and 797 DF,  p-value: < 2.2e-16

pred_lm = predict(model_train, newdata = test_data)
mse_lm = mean((pred_lm - test_data$y_)^2)
cat("Linear Regression Test MSE:", round(mse_lm, 4))
```

```
## Linear Regression Test MSE: 0.9989
```

2. KNN Regression for k = 1, 2, 5, 9, 15

```r
ks = c(1, 2, 5, 9, 15)
mse_knn = numeric(length(ks))

for (i in seq_along(ks)) {
  fit = knnreg(y_ ~ x1_ + x2_, data = train_data, k = ks[i])
  pred = predict(fit, test_data)
  mse_knn[i] = mean((test_data$y_ - pred)^2)
  cat("KNN k =", ks[i], "| Test MSE:", round(mse_knn[i], 4), "\n")
}

## KNN k = 1 | Test MSE: 2.2198
## KNN k = 2 | Test MSE: 1.7296
## KNN k = 5 | Test MSE: 1.304
## KNN k = 9 | Test MSE: 1.2054
## KNN k = 15 | Test MSE: 1.2327
```

Comparison

```r
results = data.frame(
  Method = c("Linear Regression", paste0("KNN k=", ks)),
  Test_MSE = round(c(mse_lm, mse_knn), 4)
)
print(results)

##              Method Test_MSE
## 1 Linear Regression   0.9989
## 2           KNN k=1   2.2198
## 3           KNN k=2   1.7296
## 4           KNN k=5   1.3040
## 5           KNN k=9   1.2054
## 6          KNN k=15   1.2327
```

Conclusion: Since the true data-generating process is linear, linear regression outperforms KNN. As k increases, KNN MSE decreases (bias-variance tradeoff) but still exceeds linear regression. Linear regression is the better model here when the true relationship is linear.

Nonlinear Data Generating Process

Now suppose:

$$y_i = \frac{1}{-2 + 1.4x_{1i} - 2.6x_{2i} + 2.9x_{2i}^2} + 3.1\sin(x_{2i}) - 1.5x_{1i}x_{2i}^2 + \epsilon_i$$

```r
set.seed(123)
y_nl = 1/(-2 + 1.4*x1_ - 2.6*x2_ + 2.9*x2_^2) + 3.1*sin(x2_) - 1.5*x1_*x2_^2
+ epi_
```

```r
data_nl = data.frame(y_nl, x1_, x2_)
train_nl = data_nl[1:800, ]
test_nl  = data_nl[801:1000, ]

# Linear regression
lm_nl = lm(y_nl ~ x1_ + x2_, data = train_nl)
pred_lm_nl = predict(lm_nl, newdata = test_nl)
mse_lm_nl = mean((pred_lm_nl - test_nl$y_nl)^2)
cat("Linear Regression Test MSE (nonlinear DGP):", round(mse_lm_nl, 4), "\n")
```

```
## Linear Regression Test MSE (nonlinear DGP): 162.0016
```

```r
# KNN
mse_knn_nl = numeric(length(ks))
for (i in seq_along(ks)) {
  fit = knnreg(y_nl ~ x1_ + x2_, data = train_nl, k = ks[i])
  pred = predict(fit, test_nl)
  mse_knn_nl[i] = mean((test_nl$y_nl - pred)^2)
  cat("KNN k =", ks[i], "| Test MSE:", round(mse_knn_nl[i], 4), "\n")
}
```

```
## KNN k = 1 | Test MSE: 9.4538
## KNN k = 2 | Test MSE: 10.0449
## KNN k = 5 | Test MSE: 14.9505
## KNN k = 9 | Test MSE: 17.3805
## KNN k = 15 | Test MSE: 24.8788
```

```r
results_nl = data.frame(
  Method = c("Linear Regression", paste0("KNN k=", ks)),
  Test_MSE = round(c(mse_lm_nl, mse_knn_nl), 4)
)
print(results_nl)
```

```
##              Method Test_MSE
## 1 Linear Regression 162.0016
## 2           KNN k=1   9.4538
## 3           KNN k=2  10.0449
## 4           KNN k=5  14.9505
## 5           KNN k=9  17.3805
## 6          KNN k=15  24.8788
```

Conclusion: With a nonlinear data-generating process, KNN with a small k outperforms linear regression, since KNN can capture nonlinear patterns. Linear regression has higher MSE. This demonstrates the utility of KNN over least squares when the true relationship is nonlinear.