Kernel Methods Regression

David Park

2022-10-25

```
library(e1071)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6      v purrr   0.3.4
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
## v readr   2.1.3      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
set.seed(1234)
```

Drop some problematic features and only sample 10,000 obs

```
df <- read.csv("C:\\School\\CS 4375 Machine Learning\\Kernel Ensemble Methods\\diamonds.csv")
df <- select(df, c('price', 'carat', 'depth', 'x', 'y', 'z'))
df <- sample_n(df, 10000, replace=FALSE)
head(df)
```

```
##   price carat depth    x    y    z
## 1  1168  0.61  63.4 5.37 5.43 3.42
## 2  1173  0.53  60.8 5.21 5.19 3.16
## 3   505  0.23  62.3 3.90 3.93 2.44
## 4  6118  1.33  61.3 7.11 7.08 4.35
## 5   838  0.30  61.6 4.30 4.34 2.66
## 6   911  0.30  60.8 4.34 4.31 2.63
```

Divide into train, test, validate

```
spec <- c(train=.6, test=.2, validate=.2)
i <- sample(cut(1:nrow(df), nrow(df)*cumsum(c(0,spec)), labels=names(spec)))
train <- df[i=="train",]
test <- df[i=="test",]
vald <- df[i=="validate",]
```

Explore the data

```
str(train)
```

```
## 'data.frame':    6000 obs. of  6 variables:
##  $ price: int  911 6110 612 825 2161 2587 1163 810 773 2218 ...
##  $ carat: num  0.3 1.12 0.34 0.36 0.71 0.76 0.4 0.3 0.42 0.7 ...
##  $ depth: num  60.8 61.8 59.9 59.9 64.2 61.8 62.4 63.3 60 63.9 ...
##  $ x    : num  4.34 6.64 4.57 4.62 5.62 5.88 4.74 4.3 4.85 5.57 ...
##  $ y    : num  4.31 6.7 4.55 4.66 5.66 5.84 4.72 4.23 4.88 5.51 ...
##  $ z    : num  2.63 4.12 2.73 2.77 3.62 3.62 2.95 2.7 2.92 3.54 ...
```

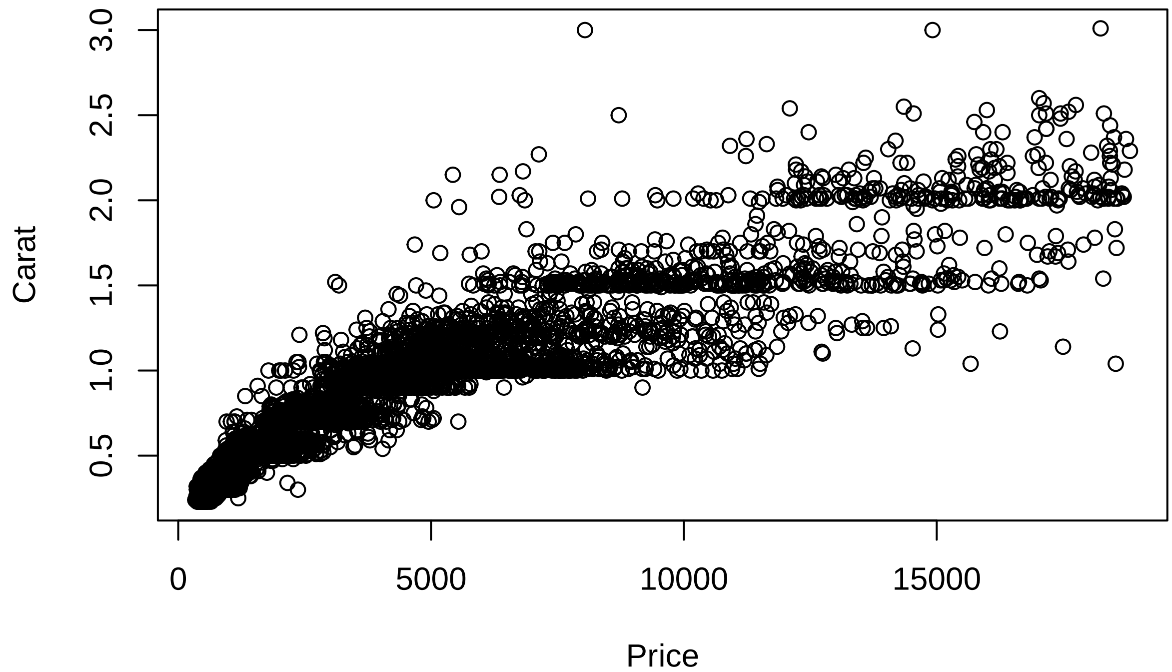
```
dim(train)
```

```
## [1] 6000    6
```

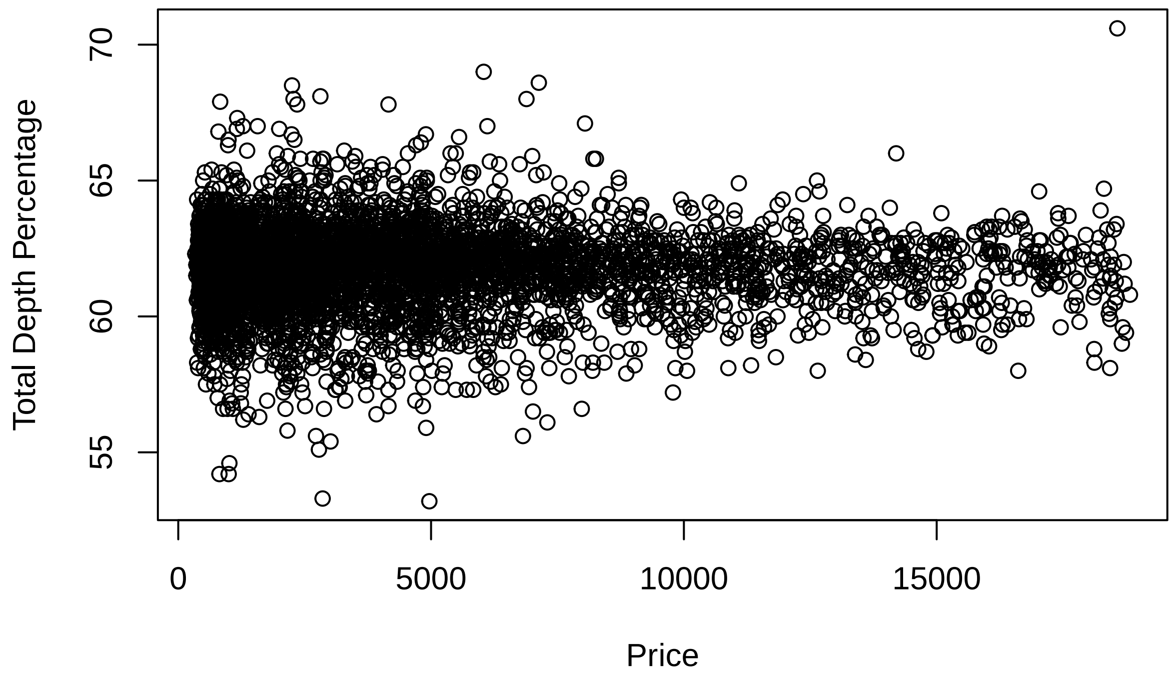
```
summary(train)
```

```
##      price             carat            depth             x        
##  Min.   :  336.0   Min.   :0.230   Min.   :53.20   Min.   :0.000  
##  1st Qu.:  939.8   1st Qu.:0.400   1st Qu.:61.00   1st Qu.:4.710  
##  Median : 2376.0   Median :0.700   Median :61.90   Median :5.690  
##  Mean   : 3899.0   Mean   :0.795   Mean   :61.74   Mean   :5.723  
##  3rd Qu.: 5257.5   3rd Qu.:1.032   3rd Qu.:62.50   3rd Qu.:6.530  
##  Max.   :18823.0   Max.   :3.010   Max.   :70.60   Max.   :9.320  
##        y               z        
##  Min.   :0.000   Min.   :0.000  
##  1st Qu.:4.720   1st Qu.:2.920  
##  Median :5.700   Median :3.520  
##  Mean   :5.726   Mean   :3.533  
##  3rd Qu.:6.510   3rd Qu.:4.030  
##  Max.   :9.190   Max.   :5.970
```

```
plot(train$price, train$carat, xlab = "Price", ylab = "Carat")
```




```
plot(train$price, train$depth, xlab = "Price", ylab = "Total Depth Percentage")
```



Linear regression for baseline

```
lm <- lm(depth~., data=train)
summary(lm)
```

```
##
## Call:
## lm(formula = depth ~ ., data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -19.281  -0.281   0.020   0.281  40.881
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  6.334e+01  1.672e-01 378.852  < 2e-16 ***
## price       -5.317e-05  7.010e-06  -7.585 3.82e-14 ***
## carat        1.538e+00  1.235e-01  12.451  < 2e-16 ***
## x           -5.698e+00  1.113e-01 -51.182  < 2e-16 ***
## y           -1.447e+00  1.025e-01 -14.116  < 2e-16 ***
## z            1.084e+01  1.005e-01 107.779  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 0.797 on 5994 degrees of freedom
## Multiple R-squared:  0.6845, Adjusted R-squared:  0.6842
## F-statistic:  2601 on 5 and 5994 DF,  p-value: < 2.2e-16
```

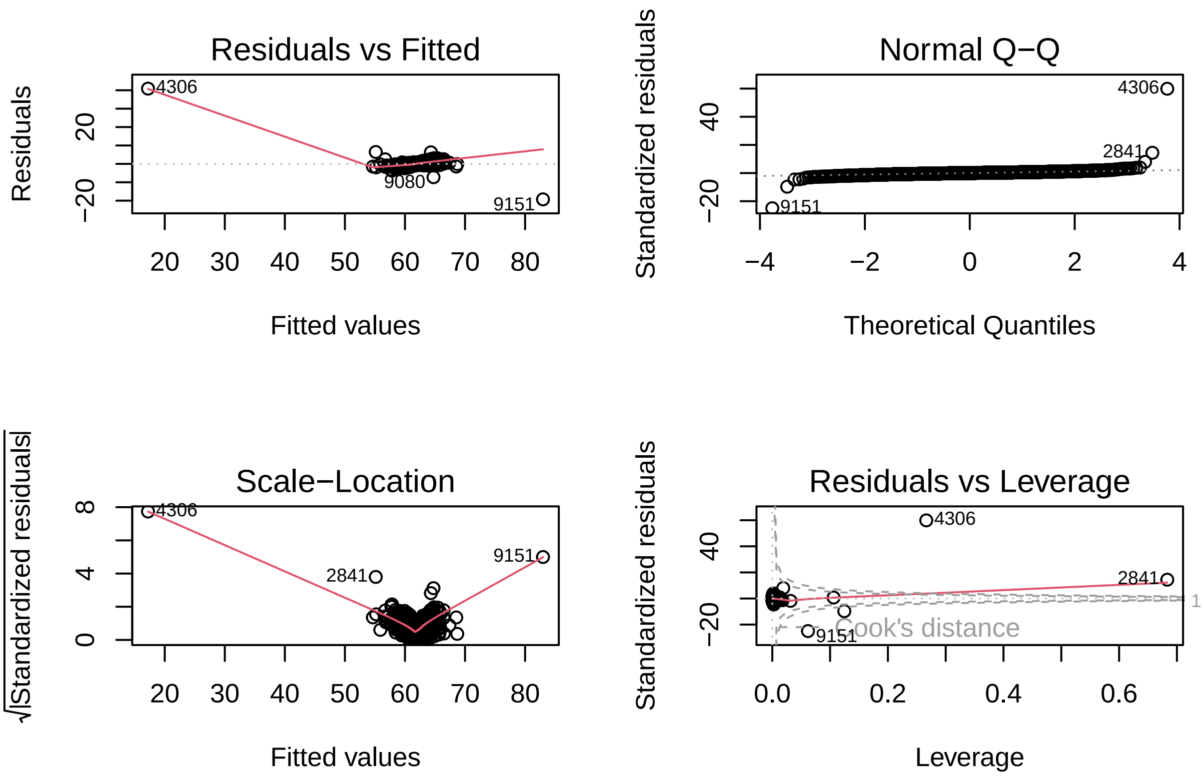
```
pred <- predict(lm, newdata=test)
cor_lm <- cor(pred, test$depth)
mse_lm <- mean((pred-test$depth)^2)
cor_lm
```

```
## [1] 0.9641417
```

```
mse_lm
```

```
## [1] 0.2678885
```

```
par(mfrow=c(2,2))
plot(lm)
```



Linear Kernel SVM Regression

```
svm_linear <- svm(depth~., data=train, kernel="linear", cost=10, scale=TRUE)
summary(svm_linear)
```

```
##
## Call:
## svm(formula = depth ~ ., data = train, kernel = "linear", cost = 10,
##     scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  linear
##        cost:  10
##       gamma:  0.2
##     epsilon:  0.1
##
##
## Number of Support Vectors:  2340
```

```r
pred <- predict(svm_linear, newdata=test)
cor_svm_linear <- cor(pred, test$depth)
mse_svm_linear <- mean((pred - test$depth)^2)
cor_svm_linear
```

```
## [1] 0.9791303
```

```r
mse_svm_linear
```

```
## [1] 0.0751517
```

Tune linear kernel SVM

```r
tune_svm_linear <- tune(svm, depth~., data=vald, kernel="linear", ranges=list(cost=c(0.001, 0.01, 0.1,
summary(tune_svm_linear)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  0.01
##
## - best performance: 1.674646
##
## - Detailed performance results:
##    cost    error dispersion
## 1 1e-03 2.026933  0.6057878
## 2 1e-02 1.674646  0.5887548
## 3 1e-01 1.794534  4.6104568
## 4 1e+00 2.388622  6.6590464
## 5 5e+00 2.467093  6.9047543
## 6 1e+01 2.475975  6.9323913
## 7 1e+02 2.488565  6.9713423
```

Evaluate linear kernel SVM with best model

```
pred <- predict(tune_svm_linear$best.model, newdata=test)
cor_svm_lin_tune <- cor(pred, test$depth)
mse_svm_lin_tune <- mean((pred - test$depth)^2)
cor_svm_lin_tune
```

```
## [1] 0.926799
```

```
mse_svm_lin_tune
```

```
## [1] 1.35003
```

Polynomial Kernel SVM Regression

```
svm_polynomial <- svm(depth~., data=train, kernel="polynomial", cost=10, scale=TRUE)
summary(svm_polynomial)
```

```
##
## Call:
## svm(formula = depth ~ ., data = train, kernel = "polynomial", cost = 10,
##     scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  polynomial
##        cost:  10
##      degree:  3
##       gamma:  0.2
##      coef.0:  0
##     epsilon:  0.1
##
##
## Number of Support Vectors:  4179
```

```
pred <- predict(svm_polynomial, newdata=test)
cor_svm_poly <- cor(pred, test$depth)
mse_svm_poly <- mean((pred - test$depth)^2)
cor_svm_poly
```

```
## [1] 0.4817662
```

```
mse_svm_poly
```

```
## [1] 1.777427
```

Tune polynomial kernel SVM

```r
tune_svm_poly <- tune(svm, depth~., data=vald, kernel="polynomial", ranges=list(cost=c(0.001, 0.01, 0.1
summary(tune_svm_poly)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 1.680677
##
## - Detailed performance results:
##    cost      error  dispersion
## 1 1e-03   2.052100   0.7049244
## 2 1e-02   1.868253   0.6765693
## 3 1e-01   1.680677   0.6993670
## 4 1e+00   2.420690   3.1612248
## 5 5e+00  13.310671  38.2546242
## 6 1e+01  23.769116  71.5669045
## 7 1e+02 150.721934 473.3374876
```

Evaluate polynomial kernel SVM

```r
pred <- predict(tune_svm_poly$best.model, newdata=test)
cor_svm_poly_tune <- cor(pred, test$depth)
mse_svm_poly_tune <- mean((pred - test$depth)^2)
cor_svm_poly_tune
```

```
## [1] 0.4576336
```

```r
mse_svm_poly_tune
```

```
## [1] 1.439193
```

Radial Kernel SVM Regression

```r
svm_radial <- svm_linear <- svm(depth~., data=train, kernel="radial", cost=10, scale=TRUE)
summary(svm_radial)
```

```
##
## Call:
## svm(formula = depth ~ ., data = train, kernel = "radial", cost = 10,
##     scale = TRUE)
##
```

```
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  10
##       gamma:  0.2
##     epsilon:  0.1
##
##
## Number of Support Vectors:  240
```

```
pred <- predict(svm_radial, newdata=test)
cor_svm_radial <- cor(pred, test$depth)
mse_svm_radial <- mean((pred - test$depth)^2)
cor_svm_radial
```

```
## [1] 0.993863
```

```
mse_svm_radial
```

```
## [1] 0.0221363
```

Tune radial kernel svm

```
tune_svm_radial <- tune(svm, depth~., data=vald, kernel="radial", ranges=list(cost=c(0.1, 1, 10, 100, 1
summary(tune_svm_radial)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##   100   0.5
##
## - best performance: 0.2277174
##
## - Detailed performance results:
##     cost gamma     error dispersion
## 1  1e-01   0.5 1.0235621  0.6512860
## 2  1e+00   0.5 0.2636217  0.6393304
## 3  1e+01   0.5 0.2297152  0.6380429
## 4  1e+02   0.5 0.2277174  0.6364043
## 5  1e+03   0.5 0.2302153  0.6353229
## 6  1e-01   1.0 0.8189205  0.6602485
## 7  1e+00   1.0 0.2691284  0.6467606
## 8  1e+01   1.0 0.2322584  0.6354249
## 9  1e+02   1.0 0.2320760  0.6338674
## 10 1e+03   1.0 0.2353216  0.6349502
```

```
## 11 1e-01   2.0 0.7098032  0.6795629
## 12 1e+00   2.0 0.2954662  0.6631850
## 13 1e+01   2.0 0.2426929  0.6288736
## 14 1e+02   2.0 0.2414817  0.6292900
## 15 1e+03   2.0 0.2421066  0.6320538
## 16 1e-01   3.0 0.6979747  0.6927631
## 17 1e+00   3.0 0.3218649  0.6738468
## 18 1e+01   3.0 0.2601245  0.6327024
## 19 1e+02   3.0 0.2590369  0.6347190
## 20 1e+03   3.0 0.2609985  0.6319589
## 21 1e-01   4.0 0.7103663  0.6996654
## 22 1e+00   4.0 0.3452584  0.6806389
## 23 1e+01   4.0 0.2808932  0.6422085
## 24 1e+02   4.0 0.2781291  0.6411058
## 25 1e+03   4.0 0.2858714  0.6396866
```

Evaluate radial kernel svm with best model

```r
pred <- predict(tune_svm_radial$best.model, newdata=test)
cor_svm_radial_tune <- cor(pred, test$depth)
mse_svm_radial_tune <- mean((pred - test$depth)^2)
cor_svm_radial_tune
```

```
## [1] 0.991785
```

```r
mse_svm_radial_tune
```

```
## [1] 0.0296982
```

How Results Were Achieved?

For my regression data, radial kernel SVM worked the best possibly due to the complexity of my data set. The data that was used would have a hard time being separated by a line. Since my data is hard to separate linearly, a higher dimension is needed, thus, the radial kernel method worked best.