SVM Regression

Charles Wallis

Data Introduction

I will be using diamonds dataset from the ggplot2 library to perform SVM regression to find the price of a diamond based on the 9 variables.

Price: price in USD ($326 - $18,823)

carat: weight of the diamond (0.2 - 5.01)

cut: cut quality (fair, good, very good, premium, ideal)

color: from D to J (Best to Worst)

Clarity: I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best)

x: length in mm

y: width in mm

z: depth in mm

depth: total depth percentage = z/mean(x,y) = 2*z/(x+y) (43-79)

table: width of top of diamond relative to widest point (43-95)

import Data

```
library(ggplot2)
data("diamonds")
data1 = diamonds
colnames(data1)
```

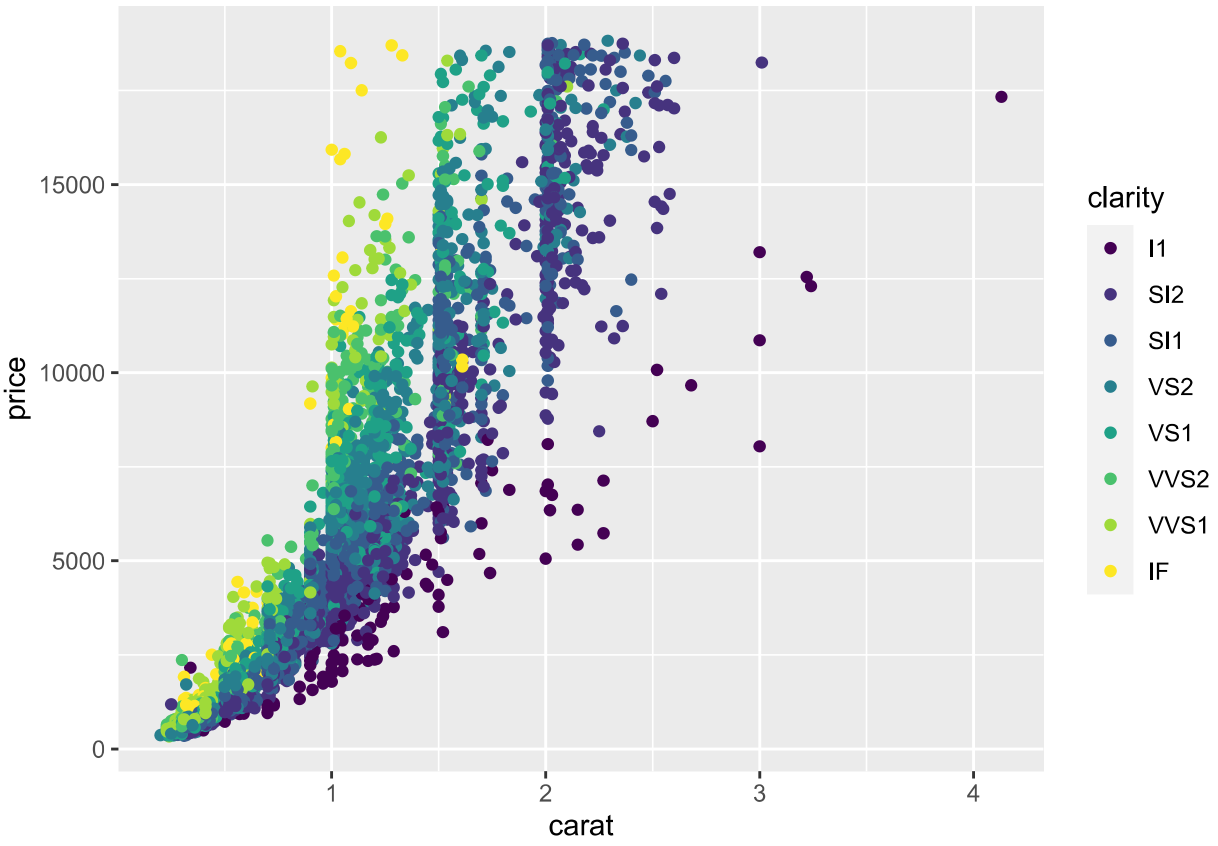
```
##  [1] "carat"   "cut"     "color"   "clarity" "depth"   "table"   "price"
##  [8] "x"       "y"       "z"
```

Cut the data size, then divide data into train/test

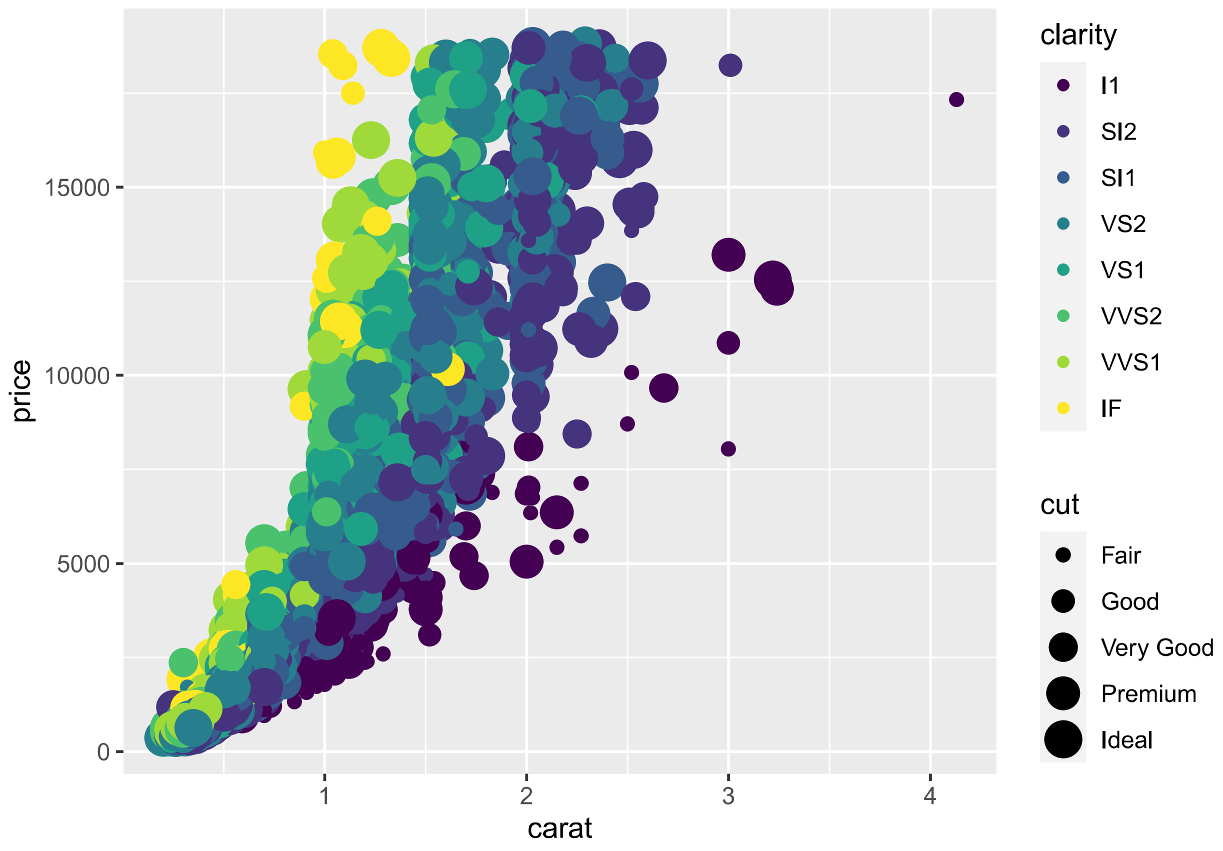
```
set.seed(1234)
cut <- sample(1:nrow(data1), nrow(data1)*0.2, replace=FALSE)
smaller <- data1[cut,]
i <- sample(1:nrow(smaller), nrow(smaller)*0.8, replace=FALSE)
train <- smaller[i,]
test <- smaller[-i,]
```

Explore training data statistics and graphs

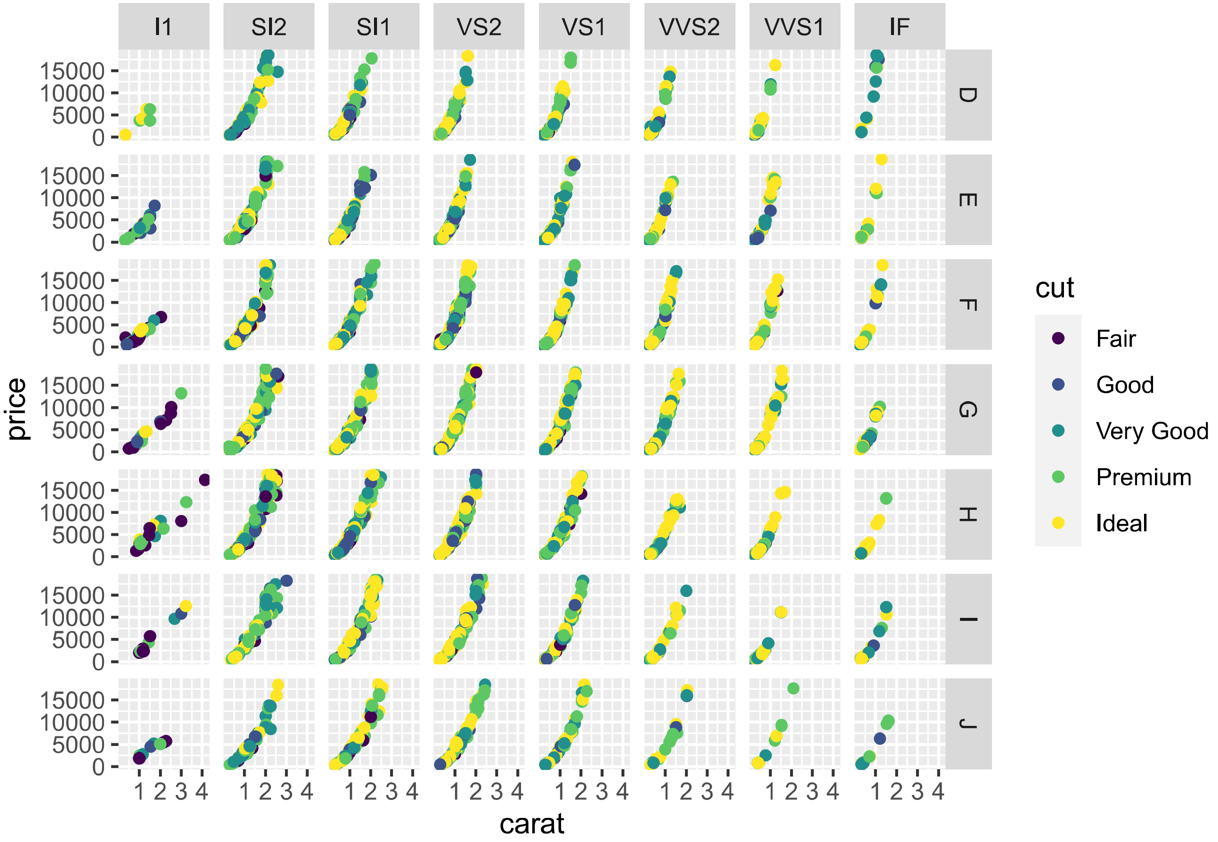
```
ggplot(train, aes(x=carat, y=price, color=clarity)) + geom_point()
```



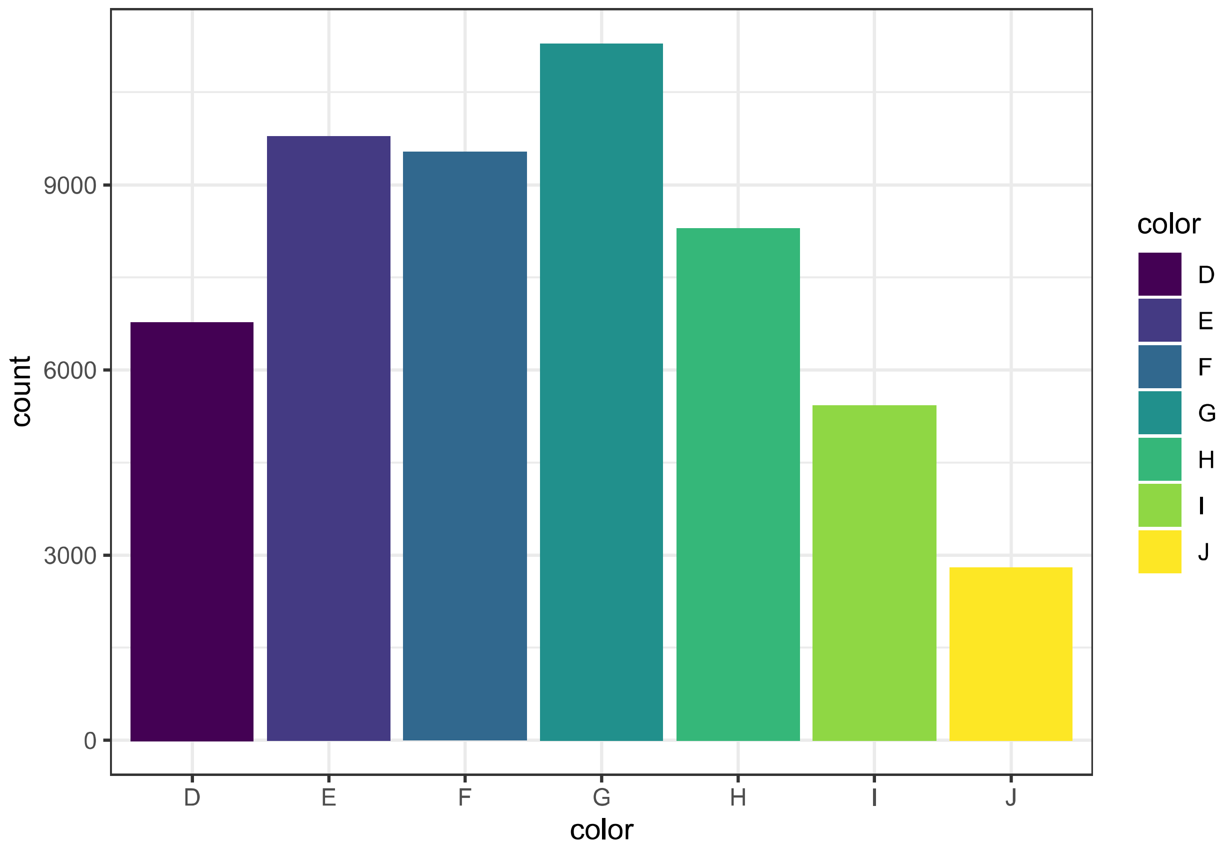
```
ggplot(train, aes(x=carat, y=price, color=clarity, size=cut)) + geom_point()
```



```
ggplot(train, aes(x=carat, y=price, color=cut)) +
  geom_point() + facet_grid(color ~ clarity)
```



```
ggplot(diamonds, aes(x = color, fill = color)) + theme_bw()+ geom_bar()
```



Perform SVM regression

```
library(e1071)
library(MASS)
svm.model <- svm(price~carat+depth+table+x+y+z, data=train)
svm.model
```

```
##
## Call:
## svm(formula = price ~ carat + depth + table + x + y + z, data = train)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.1666667
##     epsilon:  0.1
##
##
## Number of Support Vectors:  3684
```

Linear kernels with various C and gamma hyperparameters

```
svmLinear = tune(svm, price~carat+depth+table+x+y+z, data=train,
                 kernel="linear",ranges =list(cost=c(0.1, 1, 10,100)))
summary(svmLinear)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 2339433
##
## - Detailed performance results:
##    cost   error dispersion
## 1   0.1 2339433   418173.9
## 2   1.0 2361318   442388.5
## 3  10.0 2363615   445786.2
## 4 100.0 2367579   452219.2
```

```
predLinear <- predict(svmLinear$best.model, newdata=test)
cor_svmLinear <- cor(predLinear, test$price)
print(paste("Correlation: ", cor_svmLinear))
```

```
## [1] "Correlation:  0.933558026775243"
```

```
mse_linear <- mean((predLinear-test$price)^2)
print(paste("MSE: ", mse_linear))
```

```
## [1] "MSE:  2103004.82373524"
```

Polynomial kernels with various C and gamma hyperparameters

```
svmPoly = tune(svm, price~carat+depth+table+x+y+z, data=train,
               kernel="polynomial",ranges =list(cost=c(0.1, 1, 10,100)))
summary(svmPoly)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
```

```
##  cost
##   0.1
##
## - best performance: 4801464
##
## - Detailed performance results:
##    cost      error dispersion
## 1   0.1    4801464    1161742
## 2   1.0   29124422   70638166
## 3  10.0  326561672  991867789
## 4 100.0 2283788916 6875122722
```

```
predPoly <- predict(svmPoly$best.model, newdata=test)
cor_svmPoly <- cor(predPoly, test$price)
print(paste("Correlation: ", cor_svmPoly))
```

```
## [1] "Correlation:  0.48292495979177"
```

```
mse_poly <- mean((predPoly-test$price)^2)
print(paste("MSE: ", mse_poly))
```

```
## [1] "MSE:  38984373.588313"
```

Radial kernels with various C and gamma hyperparameters

```
svmRad = tune(svm, price~carat+depth+table+x+y+z, data=train,
              kernel="radial",ranges =list(cost=c(0.1, 1, 10,100)))
summary(svmRad)
```

```
##
## Parameter tuning of ’svm’:
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 1950728
##
## - Detailed performance results:
##    cost   error dispersion
## 1   0.1 2026512   242698.5
## 2   1.0 1950728   259038.0
## 3  10.0 1976104   276809.3
## 4 100.0 2116551   317434.0
```

```r
predRad <- predict(svmRad$best.model, newdata=test)
cor_svmRad <- cor(predRad, test$price)
print(paste("Correlation: ", cor_svmRad))
```

```
## [1] "Correlation:  0.944509642000985"
```

```r
mse_rad <- mean((predRad-test$price)^2)
print(paste("MSE: ", mse_rad))
```

```
## [1] "MSE:  1776085.26293374"
```

Analysis

To compare the three, the Radial kernel provided a model with the best correlation, and the polynomial kernel performed the worst with 0.48 correlation. The polynomial kernel also took the most time the compute through the training set, and the radial kernel costed the least time to compute.

In linear and polynomial, they performed the best when cost was 0.1, and for radial the performance was best when cost is 1.0.

The polynomial kernel allows to learn non-linear models, by representing similarity of vectors in the feature space to the polynomial of the original variable. It is popular in natural language processing, but it is not suitable here since our data is linear.

SVM model allows for these types of kernels so we can learn about a data set with different dimensions, to produce a readable result.

Radial kernel works like a weighted nearest neighbor model, where new data is classified based on nearby data. Cross validation determines how much weight a data will be based on distance of two points.