Capston Project: Diamonds

Johannes Le Blanc

February 10, 2019

Introduction

In this machine learning project, I analyze the diamond data set of the tidyverse package. The data set contains around 54'000 rows and ten variables. The goal is to create a model to predict the price of the diamonds.

In a first step, I load the data and delete several variables, which are not necessary for the analysis. Furthermore, I create a new variable "price_cat" from the continuous variable "price". Price_cat is a seven-step categorical variable. In a second step, I create the train and test datasets and conduct a first explorative analysis of the data by creating several plots. Subsequently, five different methods are tested on the data. Finally, the best performing method is used with the training and the test-set.

Analysis

In a first step, I install missing packages and load the data:

```
if(!require(lattice)) install.packages("lattice", repos = "http://cran.us.r-project.org")
if(!require(munsell)) install.packages("munsell", repos = "http://cran.us.r-project.org")
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")
if(!require(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-project.org")
if(!require(tidyr)) install.packages("tidyr", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(mlbench)) install.packages("mlbench", repos = "http://cran.us.r-project.org")
if(!require(rpart)) install.packages("rpart", repos = "http://cran.us.r-project.org")

data("diamonds")
head(diamonds)
```

```
## # A tibble: 6 x 10
##   carat cut       color clarity depth table price     x     y     z
##   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1 0.23  Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
## 2 0.21  Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
## 3 0.23  Good      E     VS1      56.9    65   327  4.05  4.07  2.31
## 4 0.290 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
## 5 0.31  Good      J     SI2      63.3    58   335  4.34  4.35  2.75
## 6 0.24  Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
```

```
str(diamonds)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    53940 obs. of  10 variables:
##  $ carat  : num  0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
##  $ cut    : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 3 ...
##  $ color  : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 2 5 ...
##  $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2 6 7 3 4 5 ...
##  $ depth  : num  61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
##  $ table  : num  55 61 65 58 58 57 57 55 61 61 ...
##  $ price  : int  326 326 327 334 335 336 336 337 337 338 ...
##  $ x      : num  3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
##  $ y      : num  3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
##  $ z      : num  2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
```

Clean the dataset

In this step, I delete unnecessary variables and have a first look at the data. For this analysis, I exclude all variables related to the physical dimensions of the diamonds. This includes the variables x, y, z for length, width and depth, total depth percentage, and table.

```
diamonds2 <- diamonds[c(1,2,3,4,7)]
```

I reduce the dataset of around 50'000 entrys to 10'000:

```
dataset <- diamonds2[sample(nrow(diamonds), 10000), ]
```

In this step, I convert the continuous variable price into a categorical variable of seven buckets divided by levels of $3'000:

```
dataset$price_cat <- cut(dataset$price,
                         breaks = c(0, 3000, 6000, 9000, 12000, 15000, 18000, 21000),
                         labels = c("A", "B", "C", "D", "E", "F", "G"))
```

Now I have a first look at the manipulated data:

```
head(dataset)
```

```
## # A tibble: 6 x 6
##   carat cut       color clarity price price_cat
##   <dbl> <ord>     <ord> <ord>   <int> <fct>
## 1  1.14 Ideal     H     SI1      5501 B
## 2  0.51 Very Good F     VS2      1569 A
## 3  1.53 Ideal     G     SI2      7240 C
## 4  1.7  Ideal     I     SI1     11228 D
## 5  0.69 Very Good I     VS2      2070 A
## 6  0.3  Very Good H     VVS1      878 A
```

```
summary(dataset)
```

```
##      carat                 cut        color       clarity
##  Min.   :0.2000   Fair     : 324   D:1272   SI1    :2450
##  1st Qu.:0.4000   Good     : 931   E:1806   VS2    :2358
##  Median :0.7100   Very Good:2252   F:1774   SI2    :1694
##  Mean   :0.8058   Premium  :2510   G:2104   VS1    :1466
##  3rd Qu.:1.0500   Ideal    :3983   H:1561   VVS2   : 913
##  Max.   :4.0100                    I: 966   VVS1   : 658
##                                    J: 517   (Other): 461
##      price          price_cat
##  Min.   :  336.0   A:5543
##  1st Qu.:  976.8   B:2245
##  Median : 2502.0   C:1013
##  Mean   : 3995.0   D: 546
##  3rd Qu.: 5438.2   E: 351
##  Max.   :18795.0   F: 253
##                    G:  49
```

```
dim(dataset)
```

```
## [1] 10000     6
```

Create a training and a test-set

I now split the dataset into a training and a test set. The training set contains 90% of the rows, and the test set the remaining 10%:

```
test_index <- createDataPartition(dataset$price_cat, p = 0.90, list = FALSE)
test_set <- dataset[-test_index,]
train_set <- dataset[test_index,]
```

Again, I have a quick look at the dimensions of the two new sets I have just created:

```
dim(train_set)
```

```
## [1] 9003    6
```

```
dim(test_set)
```

```
## [1] 997   6
```

Data exploration

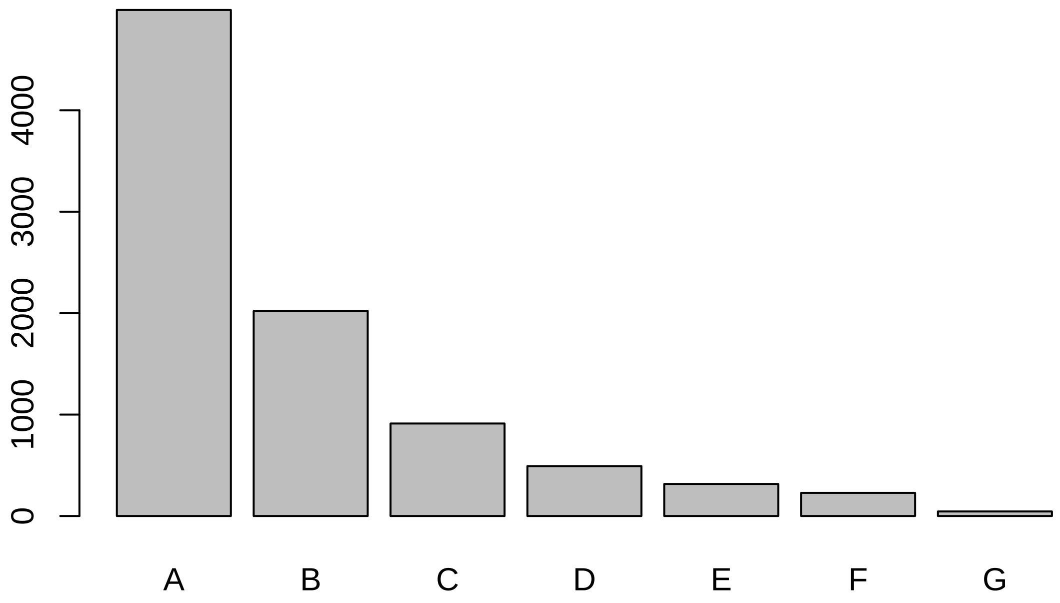
To better understand how many stones are in which price bucket, I create a table which shows the frequencies per price level:

```
price_percent <- prop.table(table(train_set$price_cat)) * 100
cbind(freq = table(train_set$price_cat), price_percent = price_percent)
```

```
##   freq price_percent
## A 4989    55.4148617
## B 2021    22.4480729
## C  912    10.1299567
## D  492     5.4648451
## E  316     3.5099411
## F  228     2.5324892
## G   45     0.4998334
```

Now, I create a simple plot of the data:

```
plot(train_set$price_cat)
```

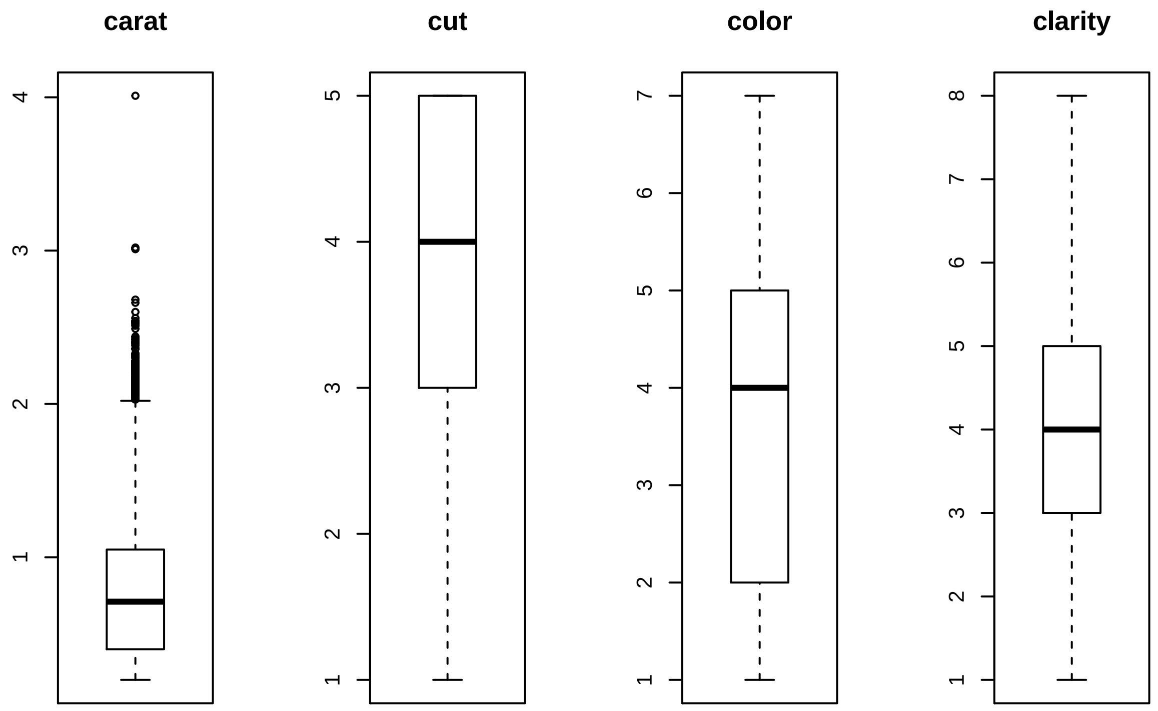


Bucket "A", which contains stones of up to $3'000 includes the vast majority of diamonds with 57% of all stones in the sample. Only 0.5% of all diamonds in the set are worth over $18'000 (bucket G). The plot visualizes the unequal distribution of stones among the buckets A-G.

Visually inspect the data

To get a better understanding of the distribution of the data, I create box-plots. For this, I create two variables, one for the output price and one for the different variables:

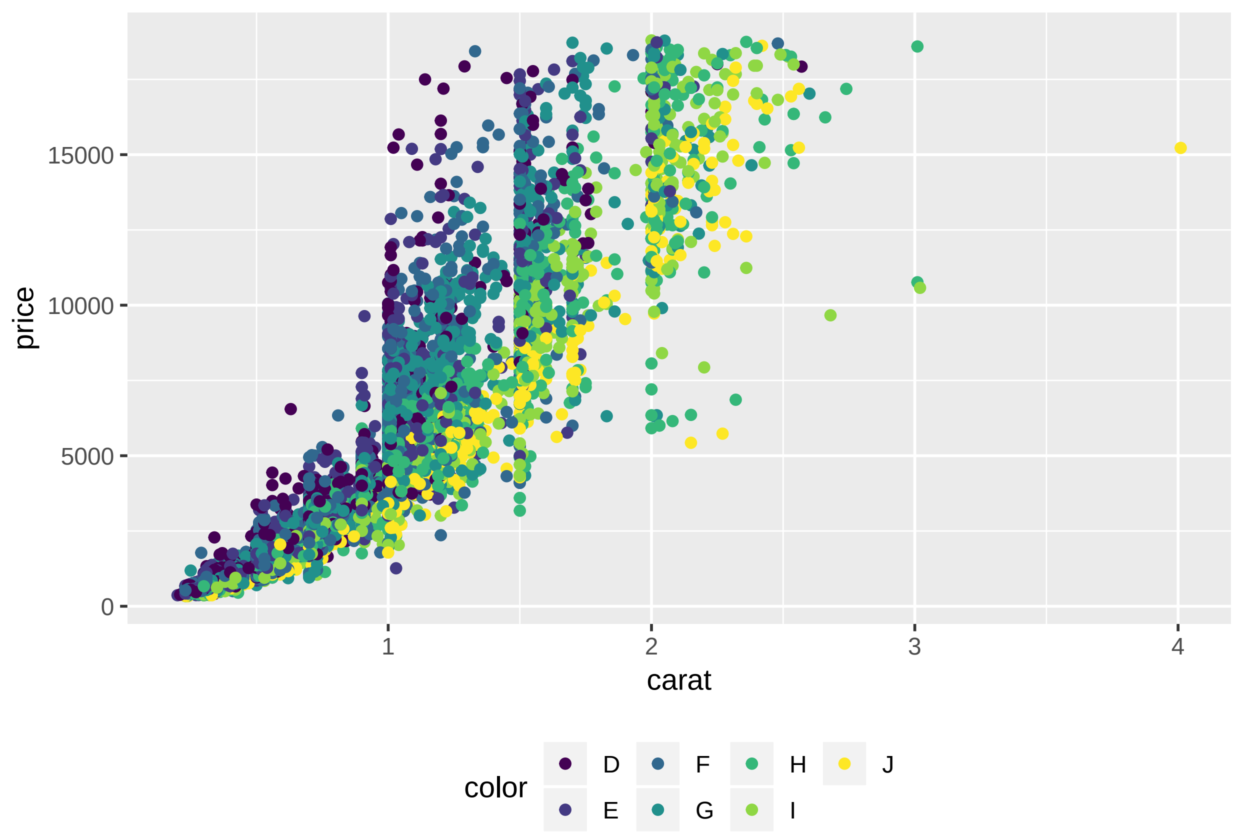
```r
x <- train_set[,1:5]
y <- train_set[,6]

par(mfrow=c(1,4))
for(i in 1:4) {
  boxplot(x[,i], main=names(train_set)[i])
}
```



The distribution of the variables differs profoundly. In particular "carat" shows a strong concentration on the lower end with several outliers going up to 4 carats. The variable "cut" in contrast, describing the quality of the cut, shows a strong accumulation at the upper end, the high-quality cuts. The medians of the two variables "color" and "clarity" again are almost located in the middle of the respective scales. The interquartile range between the median and the lower 25% percentile of "color" indicates a non-symmetric distribution with a majority of stones of low-quality color.

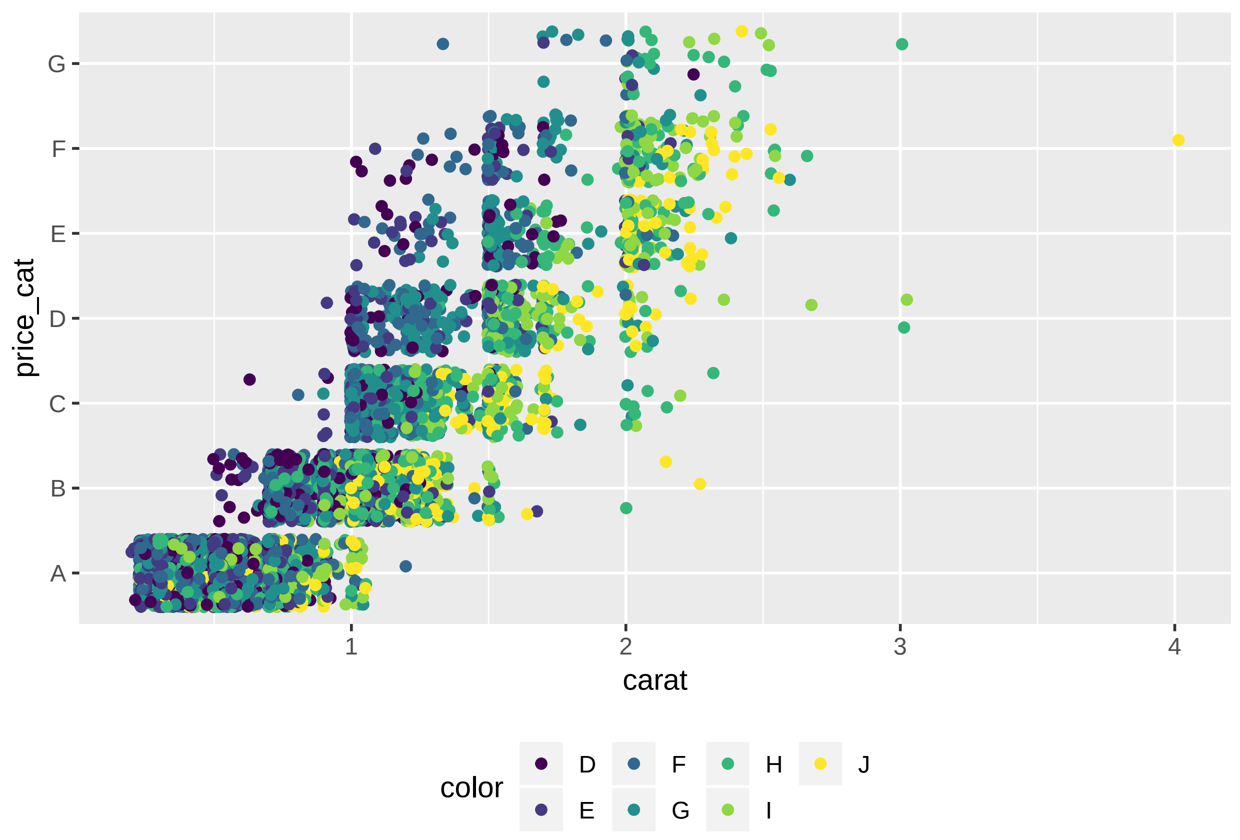
In the next step, I want to get a better understanding of the relationship between price, carat, and color. For this, I create a plot of the original data. The colors from dark-blue to yellow show the different qualities of color.

```
plot_pcc_c <- dataset %>% ggplot(aes(y = price, x = carat)) +
  geom_point(aes(color = color), alpha = 1/1)  +
  theme(legend.position="bottom")
plot_pcc_c
```



I want to explore the relationship between the variables carat and price further. For this, I plot carat on the different levels of the price buckets. For this, I now use the variable price_cat:

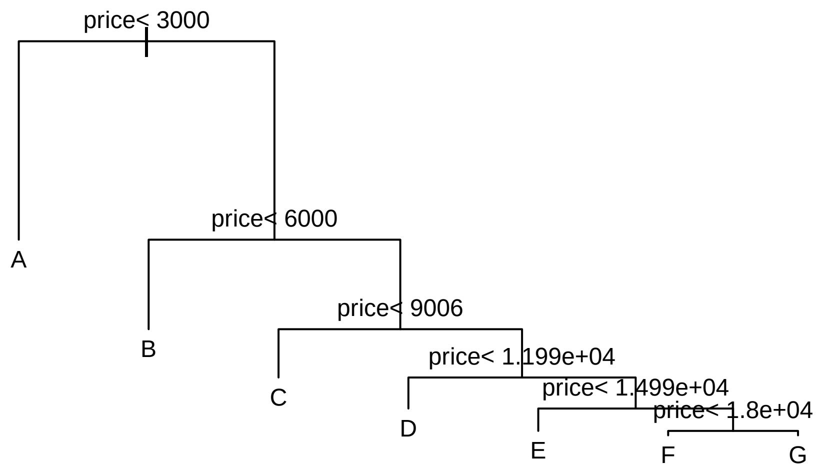
```
plot_pcc_cat <- train_set %>% ggplot(aes(y = price_cat, x = carat)) +
  geom_point(aes(color = color), position="jitter") +
  theme(legend.position="bottom")
plot_pcc_cat
```



The plot shows the different price categories on the y-axis.

In the next step, I create a regression tree to show the splits in the price variable:

```
fit_tree <- rpart(price_cat ~ ., data = train_set)
plot(fit_tree, margin = 0.1)
text(fit_tree, cex = 0.75)
```



The tree shows the seven buckets I have created before.

Creating the model

I run several algorithms and compare the outcomes. Before creating the model, I use cross-validation to make sure, our results do not come about by luck and to avoid overtraining. I start with a k = 10 cross-validation:

```
control <- trainControl(method = "cv", number = 10)
metric <- "Accuracy"
```

To find the best fitting model,I use different linear and non-linear methods to analyze the data. The methods used are:

- Linear Discriminant Analysis (LDA)
- Random Forest (RF)
- Stochastic Gradient Boosting (GBM)
- k-Nearest Neighbors (kNN)
- Support Vector Machines (SVM)

```
# LDA
set.seed(7)
fit.lda <- train(price_cat~., data = train_set, method = "lda",
                 metric = metric, trControl = control)
# Random Forest
set.seed(7)
fit.rf <- train(price_cat~., data = train_set, method = "rf",
```

```r
                 metric = metric, trControl = control)
# GBM
set.seed(7)
fit.gbm <- train(price_cat~., data = train_set, method = "gbm",
                  metric = metric, trControl = control)
# kNN
set.seed(7)
fit.knn <- train(price_cat~., data = train_set, method = "knn",
                  metric = metric, trControl = control)
# SVM
set.seed(7)
fit.svm <- train(price_cat~., data = train_set, method="svmRadial",
                  metric = metric, trControl = control)
```

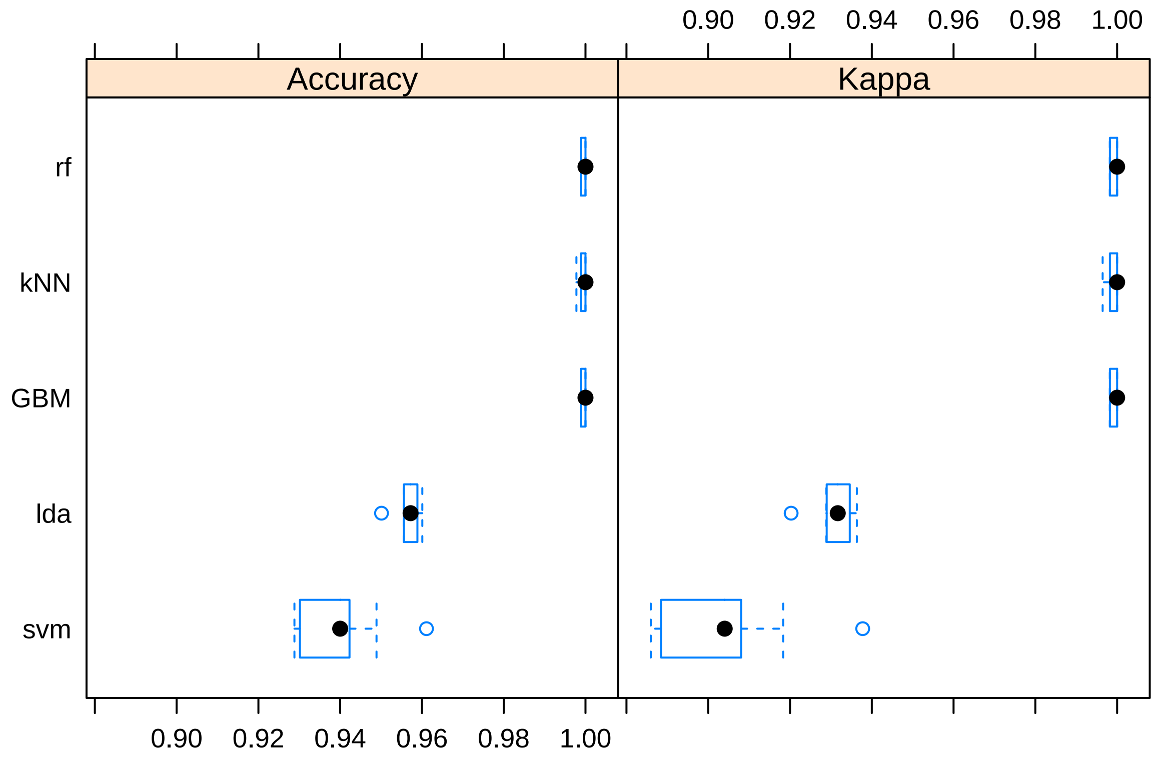
I compare the accuracy of the outcomes:

```r
results <- resamples(list(lda = fit.lda, GBM = fit.gbm, kNN = fit.knn,
                          svm = fit.svm, rf = fit.rf))

summary(results)
```

```
##
## Call:
## summary.resamples(object = results)
##
## Models: lda, GBM, kNN, svm, rf
## Number of resamples: 10
##
## Accuracy
##          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## lda 0.9501109 0.9558583 0.9572228 0.9569046 0.9586111 0.9600887    0
## GBM 0.9988877 0.9988892 1.0000000 0.9995556 1.0000000 1.0000000    0
## kNN 0.9977778 0.9988892 1.0000000 0.9993337 1.0000000 1.0000000    0
## svm 0.9288098 0.9312093 0.9400024 0.9396896 0.9422703 0.9611111    0
## rf  0.9988877 0.9988892 1.0000000 0.9995556 1.0000000 1.0000000    0
##
## Kappa
##          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## lda 0.9202745 0.9293231 0.9316719 0.9311382 0.9341151 0.9363391    0
## GBM 0.9982239 0.9982321 1.0000000 0.9992918 1.0000000 1.0000000    0
## kNN 0.9964595 0.9982320 1.0000000 0.9989388 1.0000000 1.0000000    0
## svm 0.8859454 0.8900420 0.9040239 0.9035673 0.9079072 0.9377775    0
## rf  0.9982239 0.9982321 1.0000000 0.9992918 1.0000000 1.0000000    0
```

And plot the results:

```r
bwplot(results)
```



As can be seen from the table and the plot, Random Forest, GBM and kNN do best on the diamond data, and all reach a perfect accuracy of 1. Random Forest is still slightly superior to the other two methods, why I proceed with Random Forest. Here is an overview of the results of the fitting:

```r
print(fit.rf)
```

```
## Random Forest
##
## 9003 samples
##    5 predictor
##    7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 8103, 8102, 8104, 8103, 8104, 8104, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.8935885  0.8270193
##   10    0.9994444  0.9991147
##   19    0.9995556  0.9992918
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 19.
```

Results

Finally, I test the accuracy of the result of Random Forest with the training set:

```
pred.train <- predict(fit.rf, train_set)
print(confusionMatrix(pred.train, train_set$price_cat))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E    F    G
##          A 4989    0    0    0    0    0    0
##          B    0 2021    0    0    0    0    0
##          C    0    0  912    0    0    0    0
##          D    0    0    0  492    0    0    0
##          E    0    0    0    0  316    0    0
##          F    0    0    0    0    0  228    0
##          G    0    0    0    0    0    0   45
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9996, 1)
##     No Information Rate : 0.5541
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            1.0000   1.0000   1.0000  1.00000   1.0000  1.00000
## Specificity            1.0000   1.0000   1.0000  1.00000   1.0000  1.00000
## Pos Pred Value         1.0000   1.0000   1.0000  1.00000   1.0000  1.00000
## Neg Pred Value         1.0000   1.0000   1.0000  1.00000   1.0000  1.00000
## Prevalence             0.5541   0.2245   0.1013  0.05465   0.0351  0.02532
## Detection Rate         0.5541   0.2245   0.1013  0.05465   0.0351  0.02532
## Detection Prevalence   0.5541   0.2245   0.1013  0.05465   0.0351  0.02532
## Balanced Accuracy      1.0000   1.0000   1.0000  1.00000   1.0000  1.00000
##                      Class: G
## Sensitivity          1.000000
## Specificity          1.000000
## Pos Pred Value       1.000000
## Neg Pred Value       1.000000
## Prevalence           0.004998
## Detection Rate       0.004998
## Detection Prevalence 0.004998
## Balanced Accuracy    1.000000
```

Overall accuracy, sensitivity, and specificity reach 1 for all buckets of price_cat.

And the I test the accuracy of the result of Knn with the test set:

```
pred.train <- predict(fit.rf, test_set)
print(confusionMatrix(pred.train, test_set$price_cat))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   A   B   C   D   E   F   G
##          A 554   0   0   0   0   0   0
##          B   0 224   0   0   0   0   0
##          C   0   0 101   0   0   0   0
##          D   0   0   0  54   0   0   0
##          E   0   0   0   0  35   0   0
##          F   0   0   0   0   0  25   0
##          G   0   0   0   0   0   0   4
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9963, 1)
##     No Information Rate : 0.5557
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E Class: F
## Sensitivity            1.0000   1.0000   1.0000  1.00000  1.00000  1.00000
## Specificity            1.0000   1.0000   1.0000  1.00000  1.00000  1.00000
## Pos Pred Value         1.0000   1.0000   1.0000  1.00000  1.00000  1.00000
## Neg Pred Value         1.0000   1.0000   1.0000  1.00000  1.00000  1.00000
## Prevalence             0.5557   0.2247   0.1013  0.05416  0.03511  0.02508
## Detection Rate         0.5557   0.2247   0.1013  0.05416  0.03511  0.02508
## Detection Prevalence   0.5557   0.2247   0.1013  0.05416  0.03511  0.02508
## Balanced Accuracy      1.0000   1.0000   1.0000  1.00000  1.00000  1.00000
##                      Class: G
## Sensitivity          1.000000
## Specificity          1.000000
## Pos Pred Value       1.000000
## Neg Pred Value       1.000000
## Prevalence           0.004012
## Detection Rate       0.004012
## Detection Prevalence 0.004012
## Balanced Accuracy    1.000000
```

The accuracy of the Random Forest algorithm reaches the level 1 for the test set as well.

Conclusion

In this analysis, I first loaded and restructured the data set and subsequently analyzed the data. In the data exploration section, I created several plots to visualize the distribution of the data. For the modeling section, I used five different methods to find out, which one fits the data best. Random Forest and with identical results, GBM and kNN reached the highest accuracy of all methods. I finally analyzed the accuracy of the three methods when applied to both data sets, the training set, and the test set. Random Forest reached an accuracy of 1 in both sets. Sensitivity and specificity also reached 1 for all seven buckets of the variable price_cat.

Each method used has several variants. In the next step, one could refine the analysis by running the variants of the three winning repeated with all variables of the diamonds dataset to see if any differences occur. Furthermore, one could use more methods on the data.

Bibliography

- Analytics Vidhya Content Team (2016): Practicing Machine Learning Techniques in R with MLR Package, URL: https://www.analyticsvidhya.com/blog/2016/08/practicing-machine-learning-techniques-in-r-with-mlr-package/
- ggplot2 (n.d.): Prices of 50,000 round cut diamonds, URL: https://ggplot2.tidyverse.org/reference/diamonds.html
- ggplot2 (n.d.): Points, URL: https://ggplot2.tidyverse.org/reference/geom_point.html
- Jason Brownlee (2019): Your First Machine Learning Project in R Step-By-Step, URL: https://machinelearningmastery.com/machine-learning-in-r-step-by-step/
- Joseph Rickert(2014): Comparing machine learning models in R, URL: https://blog.revolutionanalytics.com/2014/09/comparing-models-in-r.html
- Leo Breiman and Adele Cutler (n.d.): Random Forests, URL: https://www.stat.berkeley.edu/~breiman/RandomForests/
- Max Kuhn (2018): The caret Package, URL: https://topepo.github.io/caret/index.html
- Rafael A. Irizarry (2018): Introduction to Data Science: Data Analysis and Prediction Algorithms with R, URL: https://rafalab.github.io/dsbook/
- Robert I. Kabacoff (2017): Tree-Based Models, URL: https://www.statmethods.net/advstats/cart.html
- Sarajit Poddar (2015): Classification of Diamond, URL: https://rpubs.com/sarajitpoddar/diamond_classification
- Sunil Ray (2017): Essentials of Machine Learning Algorithms, URL: https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/