Goldman Sachs - Hackerrank - Car Popularity

Bala Kesavan

February 10, 2018

Context

To predict the popularity of a car given potential explanatory variables.

Exploratory data analysis

There are certain inferences we can draw about lack of popularity. Cars that are high on 'buying_price' or 'maintenance_cost' don't make the cut. Similarly, cars low on 'number_of_seats' or 'safety_rating' are not popular. 'number_of_doors' don't seem to make a difference while 'luggage_boot_size' has a weak effect on popularity.
The response variable, 'popularity', is ordinal. This has been taken care of in the pre-processing and in algorithm selection.
Next, several explanatory variables, like 'buying_price', are also ordinal. The choice was made to treat these as continuous variables after studying the well laid out arguments in this paper: Paper 248-2009, "Learning When to Be Discrete: Continuous vs. Categorical Predictors," David J. Pasta, ICON Clinical Research, San Francisco, CA.
Popularity levels 3 and 4 are under represented in the dataset. Since there are only 44 and 40 records each, a variety of subsampling techniques were tried, including SMOTE. Finally, 'upsample' was chosen since it performed best. This is understandable since SMOTE reduced the total count of data and that made it harder for the algorithm to learn.

```
library(caret)
library(ggplot2)

train_GS <- read.csv('train.csv')
summary(train_GS)
```

```
##   buying_price   maintainence_cost number_of_doors number_of_seats
##  Min.   :1.000   Min.   :1.000     Min.   :2.000   Min.   :2.000
##  1st Qu.:2.000   1st Qu.:2.000     1st Qu.:2.000   1st Qu.:2.000
##  Median :3.000   Median :3.000     Median :3.000   Median :4.000
##  Mean   :2.533   Mean   :2.528     Mean   :3.494   Mean   :3.633
##  3rd Qu.:4.000   3rd Qu.:4.000     3rd Qu.:4.250   3rd Qu.:5.000
##  Max.   :4.000   Max.   :4.000     Max.   :5.000   Max.   :5.000
##  luggage_boot_size safety_rating    popularity
##  Min.   :1.000     Min.   :1.000   Min.   :1.000
##  1st Qu.:1.000     1st Qu.:1.000   1st Qu.:1.000
##  Median :2.000     Median :2.000   Median :1.000
```

```
##  Mean   :1.987     Mean   :1.978   Mean   :1.348  
##  3rd Qu.:3.000     3rd Qu.:3.000   3rd Qu.:2.000  
##  Max.   :3.000     Max.   :3.000   Max.   :4.000  
```

```r
str(train_GS)
```

```
## 'data.frame':    1628 obs. of  7 variables:
##  $ buying_price     : int  3 3 1 4 3 2 1 3 1 2 ...
##  $ maintainence_cost: int  2 2 4 4 3 1 3 1 1 1 ...
##  $ number_of_doors  : int  4 2 2 2 3 2 5 2 3 4 ...
##  $ number_of_seats  : int  2 5 5 2 4 2 2 4 5 2 ...
##  $ luggage_boot_size: int  2 2 1 1 3 1 2 3 2 2 ...
##  $ safety_rating    : int  2 1 3 2 3 1 2 2 1 2 ...
##  $ popularity       : int  1 1 1 1 2 1 1 2 1 1 ...
```

```r
train_GS_proc <- train_GS
train_GS_proc$popularity <- as.ordered(train_GS$popularity)
str(train_GS_proc)
```

```
## 'data.frame':    1628 obs. of  7 variables:
##  $ buying_price     : int  3 3 1 4 3 2 1 3 1 2 ...
##  $ maintainence_cost: int  2 2 4 4 3 1 3 1 1 1 ...
##  $ number_of_doors  : int  4 2 2 2 3 2 5 2 3 4 ...
##  $ number_of_seats  : int  2 5 5 2 4 2 2 4 5 2 ...
##  $ luggage_boot_size: int  2 2 1 1 3 1 2 3 2 2 ...
##  $ safety_rating    : int  2 1 3 2 3 1 2 2 1 2 ...
##  $ popularity       : Ord.factor w/ 4 levels "1"<"2"<"3"<"4": 1 1 1 1 2 1
1 2 1 1 ...
```

```r
lapply(train_GS[, names(train_GS)], table)
```

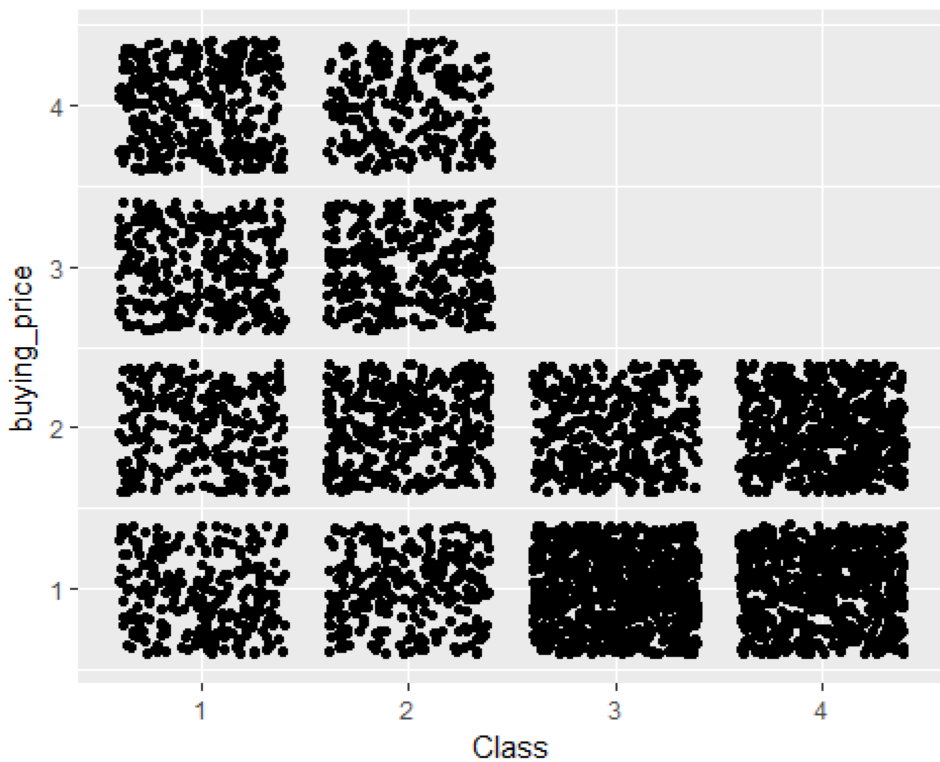
```
## $buying_price
## 
##   1   2   3   4 
## 384 408 421 415 
## 
## $maintainence_cost
## 
##   1   2   3   4 
## 392 404 412 420 
## 
## $number_of_doors
## 
##   2   3   4   5 
## 411 409 401 407 
## 
## $number_of_seats
## 
##   2   4   5 
## 565 530 533 
## 
```

```
## $luggage_boot_size
##
##   1   2   3
## 553 543 532
##
## $safety_rating
##
##   1   2   3
## 565 534 529
##
## $popularity
##
##    1    2    3    4
## 1185  359   44   40
```

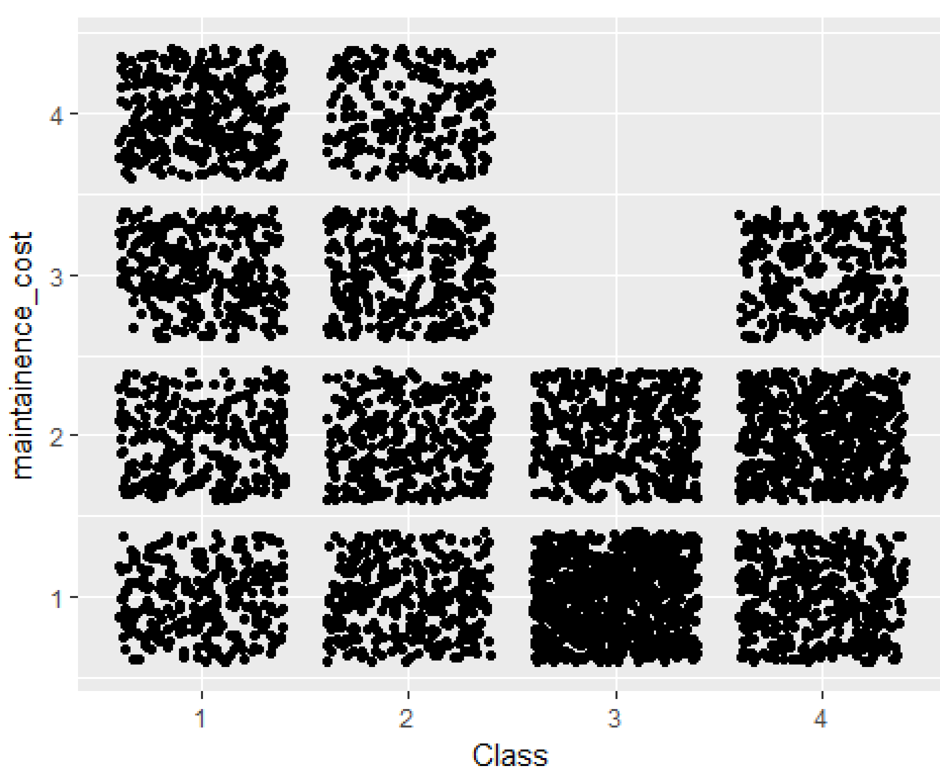
```r
#(ftable(xtabs(popularity~buying_price+maintainence_cost , data = train_GS)))

#fixing the severe under representation of popularity levels 3 & 4
train_GS_proc_upsample <- upSample(x=train_GS_proc[,-7], y=
train_GS_proc$popularity)
table(train_GS_proc_upsample$Class)
```

```
##
##    1    2    3    4
## 1185 1185 1185 1185
```

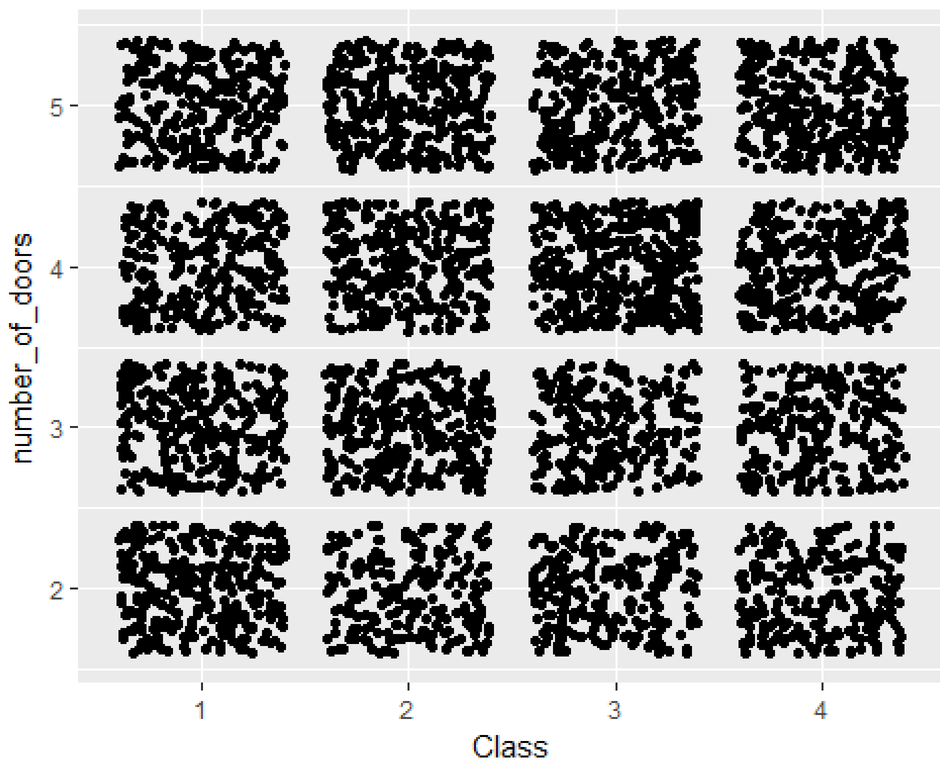
```r
#visual checks of informativeness/ ability to discriminate the response
variable
ggplot(train_GS_proc_upsample, aes(Class, buying_price)) + geom_jitter()
```



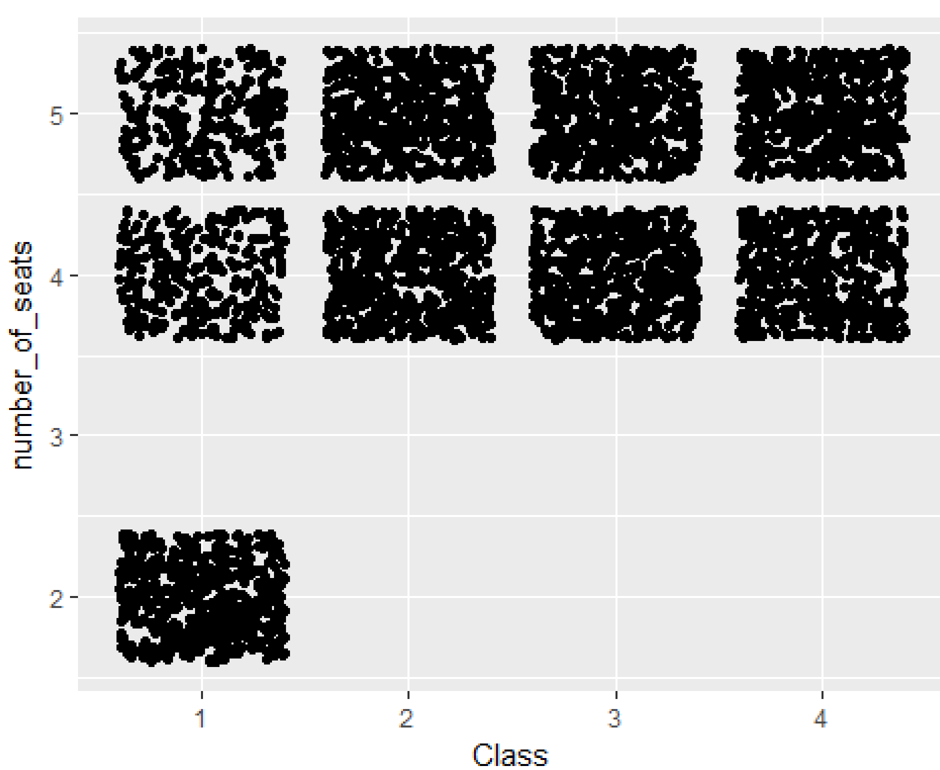

```r
ggplot(train_GS_proc_upsample, aes(Class, maintainence_cost)) + geom_jitter()
```



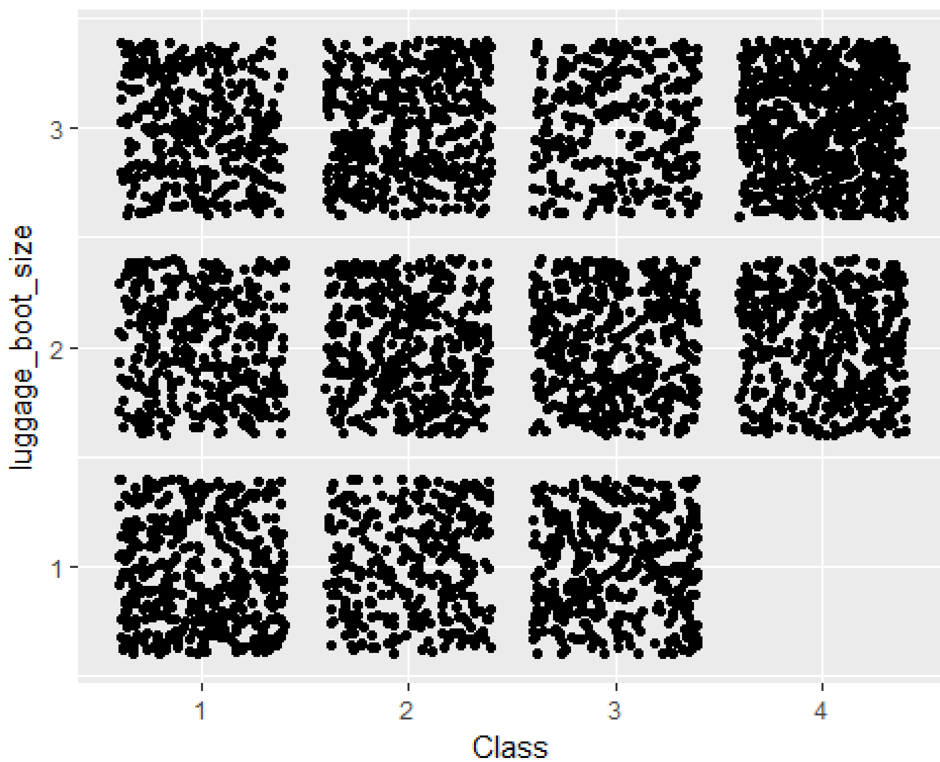

```r
ggplot(train_GS_proc_upsample, aes(Class, number_of_doors)) + geom_jitter()
```



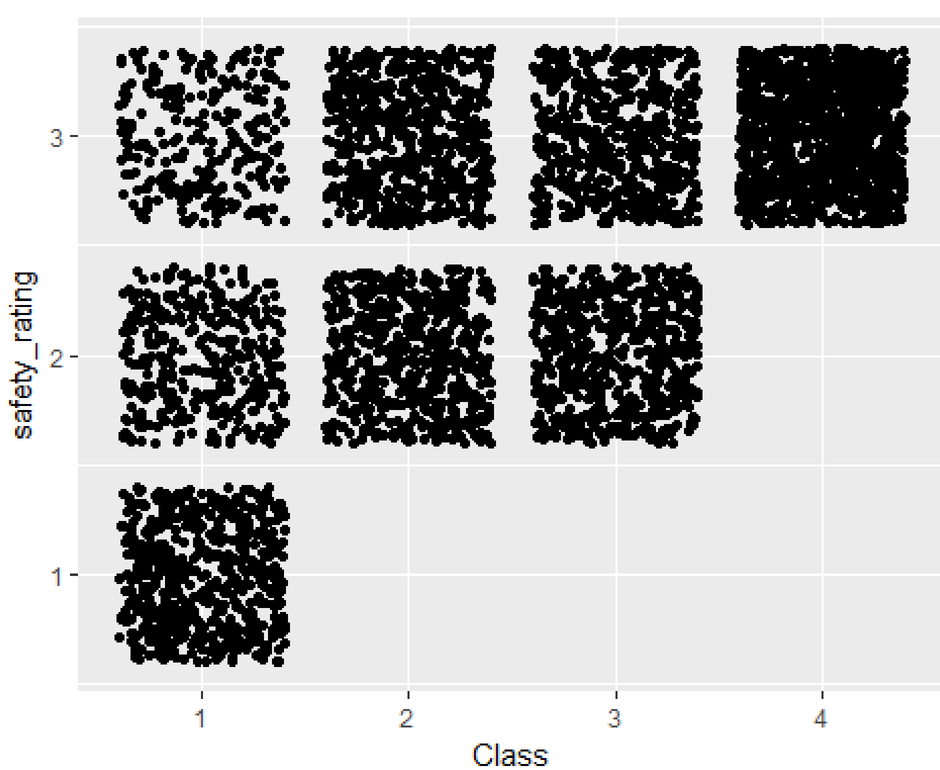

```
ggplot(train_GS_proc_upsample, aes(Class, number_of_seats)) + geom_jitter()
```



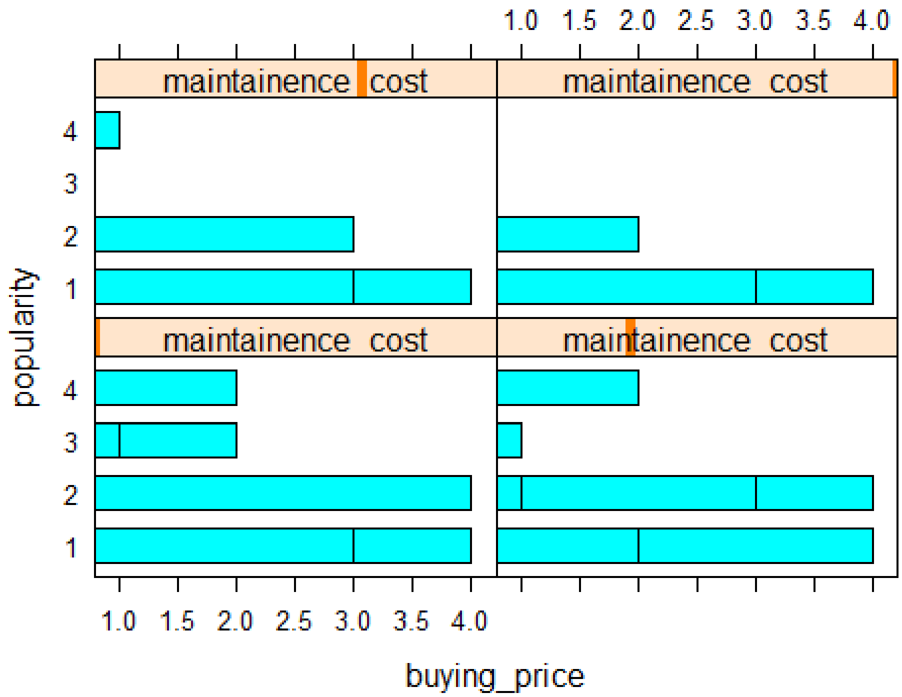

```
ggplot(train_GS_proc_upsample, aes(Class, luggage_boot_size)) + geom_jitter()
```



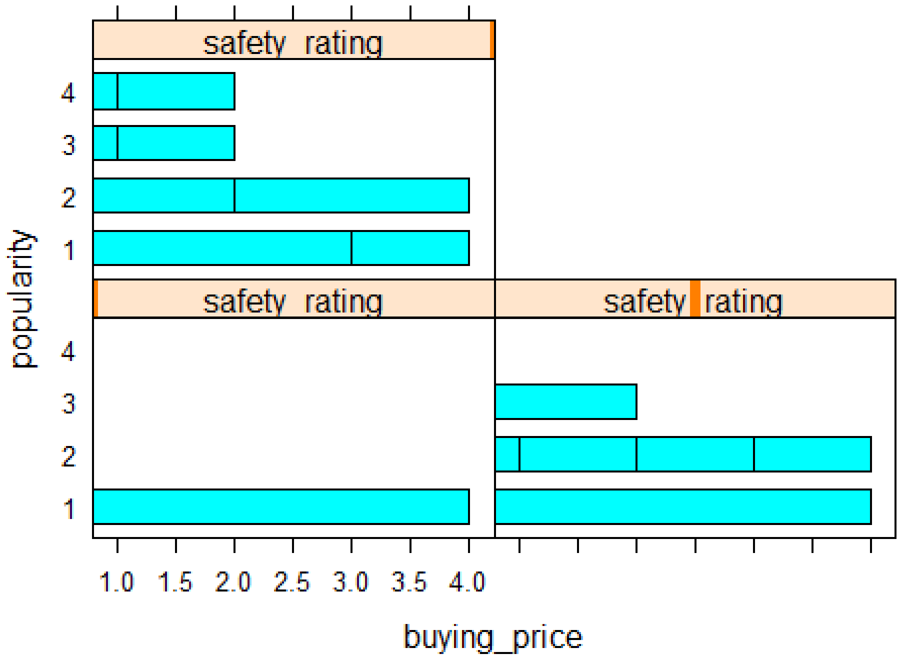

```
ggplot(train_GS_proc_upsample, aes(Class, safety_rating)) + geom_jitter()
```



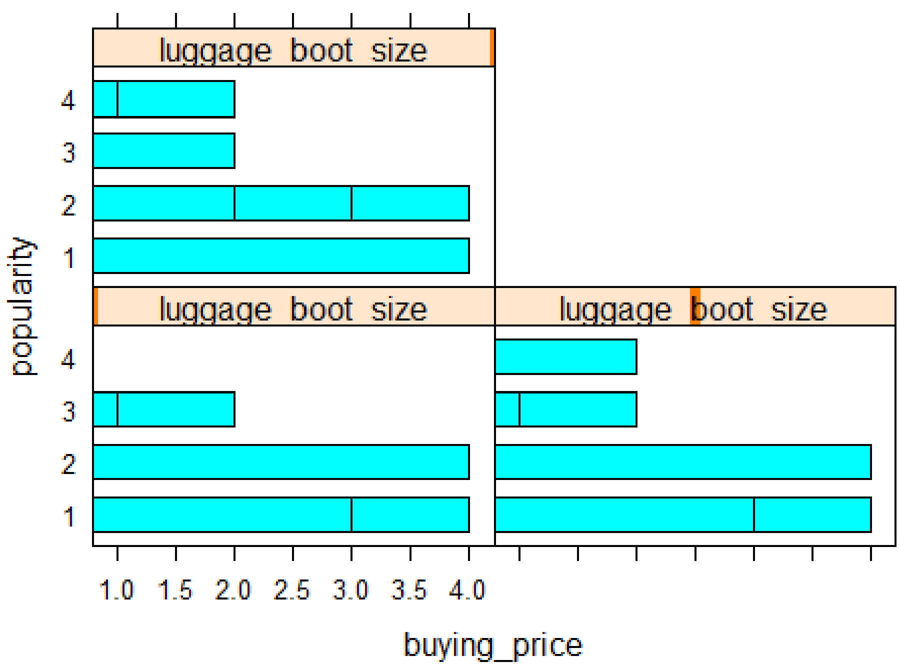
```r
barchart(Class ~ buying_price | maintainence_cost, data =
train_GS_proc_upsample,
         xlab="buying_price", ylab="popularity")
```




```r
barchart(Class ~ buying_price | safety_rating, data = train_GS_proc_upsample,
         xlab="buying_price", ylab="popularity")
```




```
barchart(Class ~ buying_price | luggage_boot_size, data =
train_GS_proc_upsample,
         xlab="buying_price", ylab="popularity")
```



```r
#satistical tests of independence
chisq.test(table(as.factor(train_GS$number_of_doors), train_GS$popularity))
```

```
##
##  Pearson's Chi-squared test
##
## data:  table(as.factor(train_GS$number_of_doors), train_GS$popularity)
## X-squared = 8.0269, df = 9, p-value = 0.5314
```

```r
chisq.test(table(as.factor(train_GS$number_of_seats), train_GS$popularity))
```

```
##
##  Pearson's Chi-squared test
##
## data:  table(as.factor(train_GS$number_of_seats), train_GS$popularity)
## X-squared = 323.68, df = 6, p-value < 2.2e-16
```

Modeling

Simple models are usually best and the 'ordinal' model with the logistic link has been used. The model coefficients are along expected lines. e.g. 'popularity' and 'buying_price' have an inverse relationship while there is a positive reeationship to 'safety_rating'.
Several options were tested, like polynomial and interaction terms. The best performing option, with one interaction term, was determined using the ANOVA test as shown below.

```r
library(ordinal)
model_clm_GS_upsample <-
clm(Class~buying_price+maintainence_cost+luggage_boot_size+safety_rating,
data = train_GS_proc_upsample)
summary(model_clm_GS_upsample)
```

```
## formula:
## Class ~ buying_price + maintainence_cost + luggage_boot_size +
safety_rating
## data:    train_GS_proc_upsample
##
##  link  threshold nobs logLik   AIC     niter max.grad cond.H
##  logit flexible  4740 -3611.70 7237.40 6(0)  1.37e-08 2.7e+03
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## buying_price      -1.77383    0.04510  -39.33   <2e-16 ***
## maintainence_cost -1.32913    0.04047  -32.84   <2e-16 ***
## luggage_boot_size  1.64197    0.04816   34.10   <2e-16 ***
## safety_rating      3.42819    0.07700   44.52   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Threshold coefficients:
```

```
##     Estimate Std. Error z value
## 1|2   2.9580     0.2034   14.54
## 2|3   6.0688     0.2154   28.17
## 3|4   8.6892     0.2370   36.66
```

```
model_clm_GS_upsample1 <- update(model_clm_GS_upsample, ~. +
buying_price*safety_rating)

anova(model_clm_GS_upsample, model_clm_GS_upsample1)
```

```
## Likelihood ratio tests of cumulative link models:
##
##                        formula:
## model_clm_GS_upsample  Class ~ buying_price + maintainence_cost +
luggage_boot_size + safety_rating
## model_clm_GS_upsample1 Class ~ buying_price + maintainence_cost +
luggage_boot_size + safety_rating + buying_price:safety_rating
##                        link: threshold:
## model_clm_GS_upsample  logit flexible
## model_clm_GS_upsample1 logit flexible
##
##                        no.par    AIC  logLik LR.stat df Pr(>Chisq)
## model_clm_GS_upsample       7 7237.4 -3611.7
## model_clm_GS_upsample1      8 7088.2 -3536.1  151.16  1  < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
test_GS <- read.csv('test.csv')
names(test_GS) <-
c("buying_price","maintainence_cost","number_of_doors","number_of_seats","lug
gage_boot_size","safety_rating")

prediction <- predict(model_clm_GS_upsample1, newdata = test_GS, type =
'class')

write.csv(prediction, file = 'prediction.csv', row.names = FALSE)
```

Conclusion

The best score obtained during submissions by Hackerrank's checker was:
F1 score = [1: 0.181818, 2: 0.260870, 3: 0.444444, 4: 0.333333]. In a way, the better performance for levels 3 and 4 is to be expected. It is easier to model the fact that cars cannot reach levels 3 or 4 of popularity without performing very well on 'buying_price', 'maintainence_cost' and 'safety_rating'.
But the reverse isn't true. Just because a car performs very well on 'buying_price', 'maintainence_cost' and 'safety_rating' doesn't lead to higher popularity.
Finding this out could be the next step.

Appendix showing confusion matrices and demonstrating the use of other algorithms

More algorithms are available in caret.

```
#data partitioning
set.seed(123)
inTraining <- createDataPartition(train_GS_proc_upsample$Class, p = 0.8, list
= FALSE)
training <- train_GS_proc_upsample[inTraining, ]
testing <- train_GS_proc_upsample[-inTraining, ]

#modeling wth CLM
library(ordinal)
#model_clm_GS_upsample <-
clm(Class~buying_price+maintainence_cost+luggage_boot_size+safety_rating,
data = training)
model_clm_GS_upsample <- readRDS('model_clm_GS_upsample.rds')
summary(model_clm_GS_upsample)

## formula:
## Class ~ buying_price + maintainence_cost + luggage_boot_size +
safety_rating
## data:    training
##
##  link  threshold nobs logLik   AIC     niter max.grad cond.H
##  logit flexible  3792 -2918.81 5851.62 6(0)  5.87e-09 2.7e+03
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## buying_price      -1.71273    0.04997  -34.27   <2e-16 ***
## maintainence_cost -1.32530    0.04477  -29.60   <2e-16 ***
## luggage_boot_size  1.65398    0.05443   30.39   <2e-16 ***
## safety_rating      3.33787    0.08454   39.48   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Threshold coefficients:
##     Estimate Std. Error z value
## 1|2   2.9690     0.2302   12.90
## 2|3   6.0286     0.2424   24.87
## 3|4   8.6054     0.2658   32.38

#saveRDS(model_clm_GS_upsample, file='model_clm_GS_upsample.rds')
#model_clm_GS_upsample1 <- update(model_clm_GS_upsample, ~. +
buying_price*safety_rating)
model_clm_GS_upsample1 <- readRDS('model_clm_GS_upsample1.rds')
summary(model_clm_GS_upsample1)
```

```
## formula:
## Class ~ buying_price + maintainence_cost + luggage_boot_size +
safety_rating + buying_price:safety_rating
## data:    training
##
##  link  threshold nobs logLik   AIC     niter max.grad cond.H
##  logit flexible  3792 -2863.11 5742.23 6(0)  5.20e-09 2.0e+04
##
## Coefficients:
##                            Estimate Std. Error z value Pr(>|z|)
## buying_price                0.05637    0.16453   0.343    0.732
## maintainence_cost          -1.35113    0.04520 -29.891   <2e-16 ***
## luggage_boot_size           1.69395    0.05525  30.659   <2e-16 ***
## safety_rating               4.71824    0.16072  29.357   <2e-16 ***
## buying_price:safety_rating -0.70803    0.06505 -10.884   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Threshold coefficients:
##     Estimate Std. Error z value
## 1|2   6.3666     0.4012   15.87
## 2|3   9.4468     0.4126   22.90
## 3|4  12.1727     0.4425   27.51

anova(model_clm_GS_upsample, model_clm_GS_upsample1)

## Likelihood ratio tests of cumulative link models:
##
##                          formula:
## model_clm_GS_upsample  Class ~ buying_price + maintainence_cost +
luggage_boot_size + safety_rating
## model_clm_GS_upsample1 Class ~ buying_price + maintainence_cost +
luggage_boot_size + safety_rating + buying_price:safety_rating
##                          link: threshold:
## model_clm_GS_upsample  logit flexible
## model_clm_GS_upsample1 logit flexible
##
##                          no.par    AIC  logLik LR.stat df Pr(>Chisq)
## model_clm_GS_upsample       7 5851.6 -2918.8
## model_clm_GS_upsample1      8 5742.2 -2863.1  111.39  1  < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

#saveRDS(model_clm_GS_upsample1, file='model_clm_GS_upsample1.rds')
prediction <- predict(model_clm_GS_upsample1, newdata = testing, type =
'class')

confusionMatrix(as.numeric(prediction$fit), as.numeric(testing$Class))

## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction   1   2   3   4
##          1 172  55   0   0
##          2  40 136  38   0
##          3  16  46 146  34
##          4   9   0  53 203
##
## Overall Statistics
##
##                Accuracy : 0.693
##                  95% CI : (0.6626, 0.7223)
##     No Information Rate : 0.25
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5907
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4
## Sensitivity            0.7257   0.5738   0.6160   0.8565
## Specificity            0.9226   0.8903   0.8650   0.9128
## Pos Pred Value         0.7577   0.6355   0.6033   0.7660
## Neg Pred Value         0.9098   0.8624   0.8711   0.9502
## Prevalence             0.2500   0.2500   0.2500   0.2500
## Detection Rate         0.1814   0.1435   0.1540   0.2141
## Detection Prevalence   0.2395   0.2257   0.2553   0.2795
## Balanced Accuracy      0.8242   0.7321   0.7405   0.8847
```

```
#modeling with 'CART or Ordinal Responses'
Control <- trainControl(method = 'cv', 10)
set.seed(21)
#model_GS_rpartScore <- train(Class~., data = train_GS_proc_upsample,
method="rpartScore", trControl=Control)
(model_GS_rpartScore <- readRDS('model_GS_rpartScore.rds'))
```

```
## CART or Ordinal Responses
##
## 4740 samples
##    6 predictor
##    4 classes: '1', '2', '3', '4'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 4267, 4266, 4267, 4266, 4267, 4267, ...
## Resampling results across tuning parameters:
##
##   cp         split  prune  Accuracy   Kappa
##   0.1136428  abs    mr     0.5647882  0.4196945
##   0.1136428  abs    mc     0.3905061  0.1873336
```

```
##   0.1136428  quad   mr     0.5261631  0.3682044
##   0.1136428  quad   mc     0.3905061  0.1873336
##   0.1477731  abs    mr     0.5261631  0.3682044
##   0.1477731  abs    mc     0.3905061  0.1873336
##   0.1477731  quad   mr     0.5261631  0.3682044
##   0.1477731  quad   mc     0.3905061  0.1873336
##   0.1589311  abs    mr     0.5261631  0.3682044
##   0.1589311  abs    mc     0.3905061  0.1873336
##   0.1589311  quad   mr     0.5261631  0.3682044
##   0.1589311  quad   mc     0.3905061  0.1873336
##
## Accuracy was used to select the optimal model using  the largest value.
## The final values used for the model were cp = 0.1136428, split = abs
##  and prune = mr.
```

```
#saveRDS(model_GS_rpartScore, file = 'model_GS_rpartScore.rds')
prediction <- predict(model_GS_rpartScore, newdata = testing)

confusionMatrix((prediction), (testing$Class))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2   3   4
##          1 139 119   0   0
##          2   0   0   0   0
##          3  79  68 128   0
##          4  19  50 109 237
##
## Overall Statistics
##
##                Accuracy : 0.5316
##                  95% CI : (0.4993, 0.5638)
##     No Information Rate : 0.25
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3755
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4
## Sensitivity            0.5865     0.00   0.5401   1.0000
## Specificity            0.8326     1.00   0.7932   0.7496
## Pos Pred Value         0.5388      NaN   0.4655   0.5711
## Neg Pred Value         0.8580     0.75   0.8380   1.0000
## Prevalence             0.2500     0.25   0.2500   0.2500
## Detection Rate         0.1466     0.00   0.1350   0.2500
## Detection Prevalence   0.2722     0.00   0.2901   0.4378
## Balanced Accuracy      0.7096     0.50   0.6667   0.8748
```

```
#modeling with 'Ordered Logistic or Probit Regression'
Control <- trainControl(method = 'cv', 10)
set.seed(21)
#model_GS_polr <- train(Class~., data = train_GS_proc_upsample,
method="polr", trControl=Control)
(model_GS_polr <- readRDS('model_GS_polr.rds'))
```

```
## Ordered Logistic or Probit Regression
##
## 4740 samples
##    6 predictor
##    4 classes: '1', '2', '3', '4'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 4265, 4266, 4267, 4266, 4266, 4264, ...
## Resampling results across tuning parameters:
##
##   method    Accuracy   Kappa
##   cauchit   0.7293173  0.6390929
##   cloglog         NaN        NaN
##   logistic  0.7248882  0.6331878
##   loglog    0.6862684  0.5816932
##   probit    0.7139114  0.6185500
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was method = cauchit.
```

```
#saveRDS(model_GS_polr, file = 'model_GS_polr.rds')

prediction <- predict(model_GS_polr, newdata = testing)

confusionMatrix((prediction), (testing$Class))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2   3   4
##          1 182  23   0   0
##          2  47 164  27   0
##          3   6  42 170  47
##          4   2   8  40 190
##
## Overall Statistics
##
##                Accuracy : 0.7447
##                  95% CI : (0.7157, 0.7722)
##     No Information Rate : 0.25
##     P-Value [Acc > NIR] : < 2.2e-16
##
```

```
##                   Kappa : 0.6596
##  Mcnemar's Test P-Value : 9.184e-05
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4
## Sensitivity            0.7679   0.6920   0.7173   0.8017
## Specificity            0.9677   0.8959   0.8664   0.9297
## Pos Pred Value         0.8878   0.6891   0.6415   0.7917
## Neg Pred Value         0.9260   0.8972   0.9019   0.9336
## Prevalence             0.2500   0.2500   0.2500   0.2500
## Detection Rate         0.1920   0.1730   0.1793   0.2004
## Detection Prevalence   0.2162   0.2511   0.2795   0.2532
## Balanced Accuracy      0.8678   0.7940   0.7918   0.8657
```

```
#modeling with 'Adjacent Categories Probability Model for Ordinal Data'
Control <- trainControl(method = 'cv', 10)
set.seed(21)
#model_GS_vglmAdjCat <- train(Class~., data = train_GS_proc_upsample,
method="vglmAdjCat", trControl=Control)
(model_GS_vglmAdjCat <- readRDS('model_GS_vglmAdjCat.rds'))
```

```
## Adjacent Categories Probability Model for Ordinal Data
##
## 4740 samples
##    6 predictor
##    4 classes: '1', '2', '3', '4'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 4265, 4266, 4267, 4266, 4266, 4264, ...
## Resampling results across tuning parameters:
##
##   parallel  Accuracy   Kappa
##   FALSE     0.8750812  0.8334416
##    TRUE     0.7250956  0.6334630
##
## Tuning parameter 'link' was held constant at a value of loge
## Accuracy was used to select the optimal model using  the largest value.
## The final values used for the model were parallel = FALSE and link = loge.
```

```
#saveRDS(model_GS_vglmAdjCat, file = 'model_GS_vglmAdjCat.rds')
prediction <- predict(model_GS_vglmAdjCat, newdata = testing)

confusionMatrix((prediction), (testing$Class))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2   3   4
##          1 200 218 157 176
```

```
##          2  25   8  80  25
##          3  12  11   0  36
##          4   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.2194
##                  95% CI : (0.1934, 0.2471)
##     No Information Rate : 0.25
##     P-Value [Acc > NIR] : 0.9874
##
##                   Kappa : -0.0408
##  Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4
## Sensitivity            0.8439 0.033755  0.00000     0.00
## Specificity            0.2250 0.817159  0.91702     1.00
## Pos Pred Value         0.2663 0.057971  0.00000      NaN
## Neg Pred Value         0.8122 0.717284  0.73341     0.75
## Prevalence             0.2500 0.250000  0.25000     0.25
## Detection Rate         0.2110 0.008439  0.00000     0.00
## Detection Prevalence   0.7922 0.145570  0.06224     0.00
## Balanced Accuracy      0.5345 0.425457  0.45851     0.50
```