CYO Project: Car classification

Dr. Ankur Awadhiya, IFS

15/08/2021

Contents

Executive summary

Machine learning is a branch of computer science and data analytics that automates the construction and deployment of analytical models. It trains algorithms on training datasets, tests them on testing datasets, and then deploys them in world situations. The large availability of data and faster processing power has meant that more and more of classification and decision-making tasks are now being done through the use of machine learning, including, but not limited to automatic reading of optical characters such as zip codes and book scans, self-driving vehicles, speech recognition, and targeted advertising.

In this project, we aim to utilise machine learning for the task of classification of vehicles. Such an activity is important for, say, a used car dealership that needs to assign a dollar value to the used vehicles, based on their attributes. Today most of such decisions are made by humans who look at the vehicle and consider their attributes to classify the vehicles into categories such as very good, good, acceptable, and unacceptable. Our aim is to train machine learning algorithms to do the job.

To this end, we make use of a public "Car evaluation dataset" from the University of California, Irvine's Machine learning repository. This is a multivariate dataset with categorical attributes. We perform cleaning and visualisation of the dataset, and construct decision trees.

To train several machine learning algorithms on the dataset, we first partition the dataset into working and validation datasets, and then sub-partition the working dataset into train and test datasets. Several machine learning algorithms are trained on the train dataset and tested on the test dataset. We consider the accuracy and other parameters to choose the best algorithm for the task.

The selected algorithm (svmRadial) is then validated on the validation dataset. We find that the selected algorithm is able to make correct predictions on the validation dataset with an accuracy of greater than 95%, perhaps much better than what a human would have achieved.

The speed, reliability, accuracy, cost-efficiency and robustness of machine learning as evident from this project demonstrates the utility of machine learning in automating classification and prediction tasks. We also comment on the limitations and future works in appropriate sections.

Introduction

Classification is an important activity for several purposes. We classify objects around us to make sense of the world. Doctors classify diseases to quickly make a diagnosis and to hasten treatment. In fact, classification has been ingrained in all of us through evolution. There are many evidences of classification done by several species of animals as well.

In this project, we explore classification of vehicles through employment of machine learning. Classification of vehicles into different categories is an extremely useful task for firms dealing with sale of used cars. If we can classify vehicles into grades such as unacceptable, acceptable, good, and very good, we can make decisions about, say, how much to charge for the vehicles, how to upkeep the vehicles to improve their categories, and so on.

This categorisation has traditionally been done by humans. We take several objective and subjective parameters - the buying cost of the vehicle (a used Mercedes will probably sell for a lot more than a used Honda), the class of the vehicle (a sports vehicle or an SUV will probably sell for much more than a hatchback), the yearly maintenance cost of the vehicle (a vehicle requiring extensive maintenance is probably not in a very good condition), how much the vehicle has been driven (the more the odometer reading, the less valuable is the vehicle), the look of the vehicle (a vehicle appearing dilapidated will sell for much less than a good looking vehicle), and so on.

With these parameters, we can make an assessment of the categorisation of the vehicle in question. Often to increase objectivity, defined rules are used by sales companies, wherein points are awarded for several objective parameters. However, this often creates a new categorisation - probably very different from the older one that used

both objective and subjective parameters. And in case there is a merger of companies, old stocks will need to be categorised again to add to the stock. To do such activities quickly and efficiently, we can make use of machine learning.

Goals

In this exercise, we take a dataset with vehicle parameters and classifications. The goal is to train several machine learning algorithms to predict these classifications. We test the trained algorithms using another portion of the dataset (called the test dataset), and select the best-performing algorithm. This algorithm will then be validated on an untouched portion of the dataset (called the validation dataset). When successful, such a trained algorithm can be used on new data for automatic classification of vehicles.

Dataset

The dataset used in this project is the "Car evaluation dataset" from the University of California, Irvine's Machine learning repository. This is a multivariate dataset with categorical attributes. There are 1,728 instances with 6 attributes:

1. buying price
2. price of maintenance
3. number of doors
4. person capacity
5. size of luggage boot
6. safety of the car

On the basis of these attributes, the cars are classified into 4 classes:

1. unacceptable
2. acceptable
3. good
4. very good

Variables

The independent variables and their values are:

1. buying cost aka buying: vhigh, high, med, low
2. maintenance cost aka maint: vhigh, high, med, low
3. doors: 2, 3, 4, 5more (representing 5 or more)
4. persons: 2, 4, more
5. size of luggage boot aka lug_boot: small, med, big
6. safety: low, med, high

The dependent variable is class value, which takes the values: unacc, acc, good, vgood (representing unacceptable, acceptable, good and very good).

Key steps

The key steps are:

1. Downloading the dataset.

2. Cleaning the dataset to remove NA values to facilitate analysis.

3. Exploration and visualisation of dataset to make sense of it.

4. Dividing the dataset into working and validation datasets. All the algorithms are trained and tested on the working dataset. The validation dataset is kept untouched till the very end.

   Such a division ensures that our final results are not overly cherry picked. Since we train and test algorithms on the working dataset only, any decisions made, such as selection of algorithm(s) or their parameter(s) are made without any knowledge of the validation dataset. In this case, when the final validation of the selected algorithm(s) and parameter(s) is done using the validation dataset, we can be sure that the results we obtain are not overly optimistic, and are in line with what the algorithms will face out in the world when they get deployed.

   Since the employed dataset is a large dataset, we take a 90:10 division so that we have sufficient instances to train and test, while also retaining sufficient instances to validate. This division is done randomly using random partitioning algorithm.

5. Dividing the working portion of the dataset into train and test datasets. All the algorithm(s) and parameter(s) are trained on the train dataset, and their accuracy is tested on the test dataset.

   Such a division ensures that we have reliable testing results. If the algorithms were tested on the same dataset on which they were trained, they would have overly optimistic accuracies.

   Since the employed dataset is a large dataset, we take a 90:10 division so that we have sufficient instances to train and test. This division is done randomly using random partitioning algorithm.

6. Making a decision tree. Decision trees provide an intuitive understanding of classification by creating a set of questions about attributes whose answering in "yes" or "no" lead to a bifurcated branch of the tree. By answering several "yes" and "no" questions, we are led to a final categorisation of the target. For example, a vehicle that costs less, requires less maintenance cost, while providing good safety, comfort and technology will be a very good vehicle. On the other hand, a vehicle that costs much, requires large maintenance cost, while providing unacceptable safety, comfort and technology will be a very bad, or unacceptable, vehicle. Decision trees provide a way to objectively make these decisions using "yes-no" answers.

7. Training and testing algorithms. There are several machine learning algorithms that can be used for prediction and categorisation. To make a selection about the most optimal algorithm, we train several algorithms on the train dataset, and then test them on the test dataset. The one(s) that provide the best accuracy are then selected.

8. Validation. The selected algorithm(s) are validated on the validation dataset. To do this, we train the selected algorithm(s) on the complete working dataset - this training on a larger dataset than the train dataset is done to provide a better training to the algorithm. Then the trained algorithm is tested on the validation dataset to check how good it would perform on a hitherto unknown dataset. If the trained algorithm is able to provide an acceptable level of accuracy on the validation dataset (say, above 90%), we'll say that machine learning can be relied upon to do the work.

Methods and analyses

Basic options

Before we begin our analysis, we clear the existing variables in the workspace using the rm(list=ls()) command. We also set the number of decimal places for results using the options(digits=4) command.

```
### Basic options: clear list, show 4 decimal places
rm(list=ls())
options(digits=4)
```

Installing requisite packages

When submitting a code to a third party, it is a good practice to incorporate commands that install the requisite packages automatically, so that the code runs smoothly on the target machine. This is done using the if(!require(PACKAGE_NAME)) install.packages("PACKAGE_NAME", repos = "REPO_NAME") command. This command checks if the requisite packages are installed on the target machine, and if not, installs them. We make a list of requisite packages and incorporate the installation commands.

```
### Installing required packages
if(!require(tidyverse)) install.packages("tidyverse",
    repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret",
    repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table",
```

```
    repos = "http://cran.us.r-project.org")
if(!require(rpart)) install.packages("rpart",
    repos = "http://cran.us.r-project.org")
if(!require(rpart.plot)) install.packages("rpart.plot",
    repos = "http://cran.us.r-project.org")
if(!require(HDclassif)) install.packages("HDclassif",
    repos = "http://cran.us.r-project.org")
if(!require(naivebayes)) install.packages("naivebayes",
    repos = "http://cran.us.r-project.org")
if(!require(C50)) install.packages("C50",
    repos = "http://cran.us.r-project.org")
if(!require(e1071)) install.packages("e1071",
    repos = "http://cran.us.r-project.org")
if(!require(ipred)) install.packages("ipred",
    repos = "http://cran.us.r-project.org")
if(!require(plyr)) install.packages("plyr",
    repos = "http://cran.us.r-project.org")
if(!require(gam)) install.packages("gam",
    repos = "http://cran.us.r-project.org")
if(!require(kernlab)) install.packages("kernlab",
    repos = "http://cran.us.r-project.org")
if(!require(randomForest)) install.packages("randomForest",
    repos = "http://cran.us.r-project.org")
```

Loading requisite libraries

Next, we load the requisite package libraries using the library(LIB_NAME) command. Once the libraries are loaded, we can make use of them in the subsequent code.

```
### Loading required libraries
library(plyr)
library(tidyverse)
library(caret)
library(data.table)
library(rpart)
library(rpart.plot)
library(HDclassif)
library(naivebayes)
library(C50)
library(e1071)
library(ipred)
library(gam)
library(kernlab)
library(randomForest)
```

Dataset download

It is a good practice to incorporate the commands for downloading the dataset in the script itself, so that the script can run autonomously. In the present case, the dataset is to be downloaded from the University of California, Irvine's Machine learning

repository. The dataset is in the form of comma-separated values, and we load the dataset using the read.csv() command.

```
### Downloading the dataset
link <- "https://archive.ics.uci.edu/ml/machine-learning-databases/car/car.data"
db = read.csv(link)
```

Data cleaning

Often the data has NA values, or is missing the header row. At times, the ordinal data need to be re-factored as per requirement. All these operations are done in the data cleaning stage.

We begin by inspecting the database. The head() command shows the first 6 rows of the dataset, and the dim() command reveals the dimensions (number of rows and columns) of the dataset.

```
### Inspecting the dataset
head(db) # Look at the first 6 rows
```

```
##   vhigh vhigh.1 X2 X2.1 small  low unacc
## 1 vhigh   vhigh  2    2 small  med unacc
## 2 vhigh   vhigh  2    2 small high unacc
## 3 vhigh   vhigh  2    2   med  low unacc
## 4 vhigh   vhigh  2    2   med  med unacc
## 5 vhigh   vhigh  2    2   med high unacc
## 6 vhigh   vhigh  2    2   big  low unacc
```

```
dim(db) # Look at the dataset dimensions
```

```
## [1] 1727    7
```

We observe that our dataset is missing the header row. The dimensions are 1727 rows and 7 columns. We thus add the column names using the colnames() command.

```
### Adding column names to the dataset
colnames(db)=c("buying","maint","doors","persons","lug_boot",
    "safety","class")
```

While our dataset does not have NA values, it is a good practice to incorporate removal of NA values during processing of any dataset, if just for maintaining a habit. We can remove the rows with NA values using the db[complete.cases(db),] code. The complete.cases() command returns those rows with complete values (i.e., no NA values). We place all the data in complete rows in the variable clean:

```
### Cleaning data by removing rows with NA values
clean <- db[complete.cases(db),]
```

We next inspect the cleaned dataset using the head() and dim() commands. In this case, we make use of the pipe %>% command, provided by the tidyverse library.

```
### Inspecting the cleaned dataset
clean %>% head() # Look at the first 6 rows
```

```
##   buying maint doors persons lug_boot safety class
```

```
## 1  vhigh vhigh     2       2    small    med unacc
## 2  vhigh vhigh     2       2    small   high unacc
## 3  vhigh vhigh     2       2      med    low unacc
## 4  vhigh vhigh     2       2      med    med unacc
## 5  vhigh vhigh     2       2      med   high unacc
## 6  vhigh vhigh     2       2      big    low unacc
```

```
dim(clean) # Look at the cleaned dataset dimensions
```

```
## [1] 1727    7
```

We can observe that the cleaned dataset has header rows, and comprises 1727 rows and 7 columns. Since the number of rows is the same as before, we know that there were no NA values in the raw dataset. We next provide correct factors to the ordinal data using the factor() command:

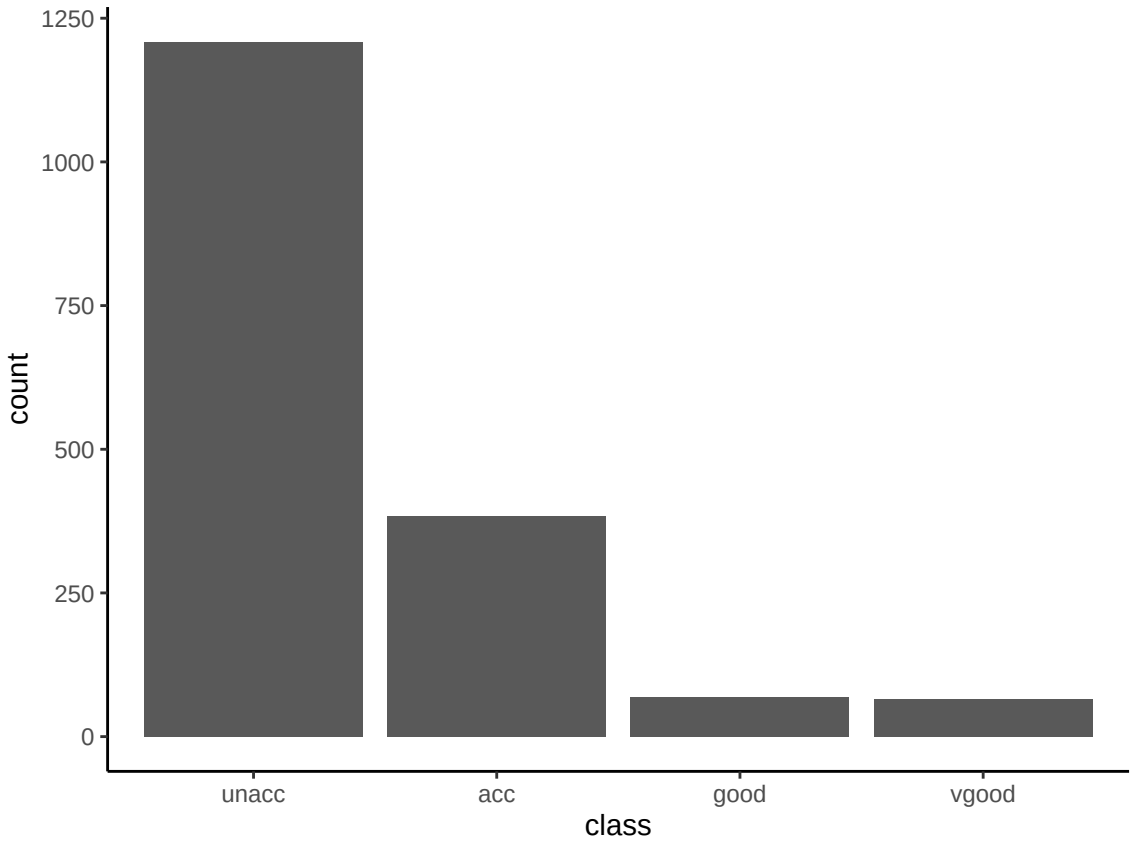
```
### Rearrange dataset values to improve graph visualisation
clean$buying <- factor(clean$buying,levels = c("low", "med",
                "high", "vhigh"))
clean$maint <- factor(clean$maint,levels = c("low", "med",
                "high", "vhigh"))
clean$class <- factor(clean$class,levels = c("unacc", "acc",
                "good", "vgood"))
clean$safety <- factor(clean$safety,levels = c("low", "med",
                "high"))
clean$lug_boot <- factor(clean$lug_boot,levels = c("small",
                "med", "big"))
```

This completes the cleaning stage.

Data exploration and visualisation

To make sense of the dataset, we can plot histograms. We first plot the vehicle classes on the X-axis, and the number of vehicles on the Y-axis.

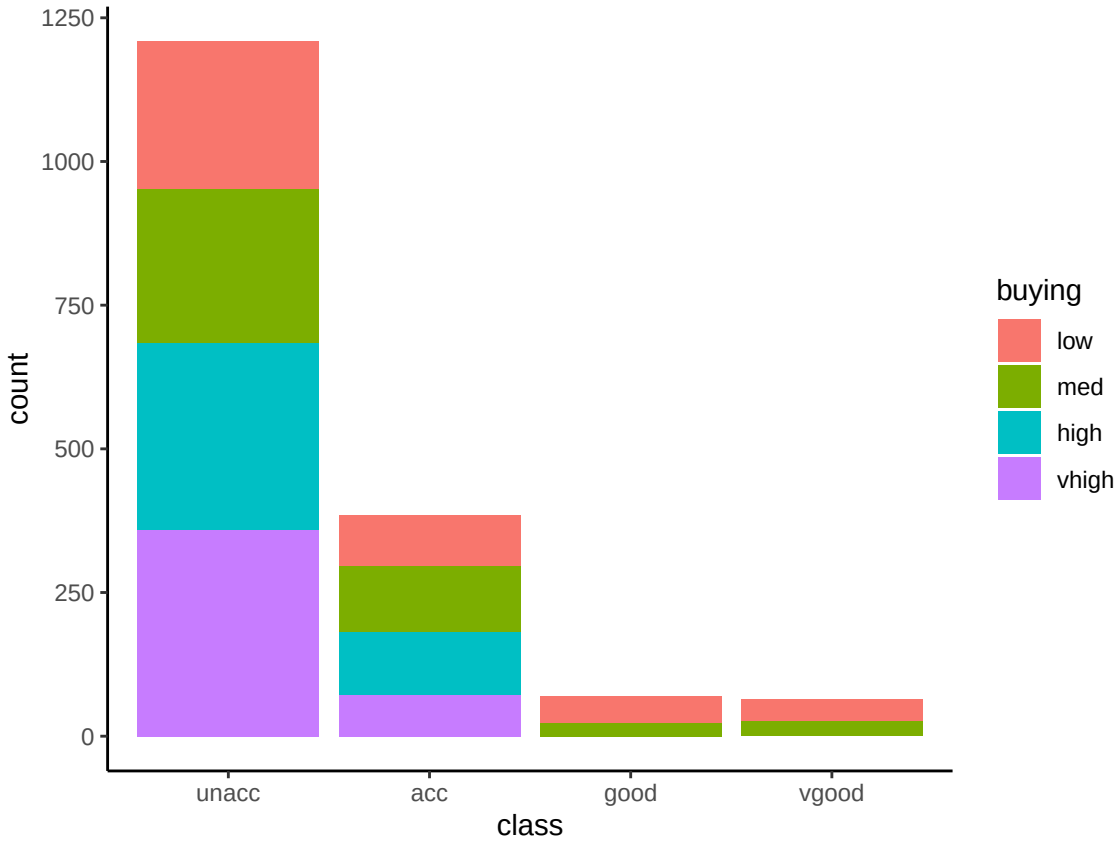
```
### Data exploration and visualisation

### Class vs. count, filled with attributes

clean %>% ggplot(aes(x=class))+geom_histogram(stat="count")+

            theme_classic()
```



We can observe that most of the vehicles are of unacceptable category, followed by acceptable and good, and very few vehicles are of very good category. Thus, the dataset is skewed. We can now explore the parameters that determine these categories. We

plot the histogram, but now with colours given by buying costs:

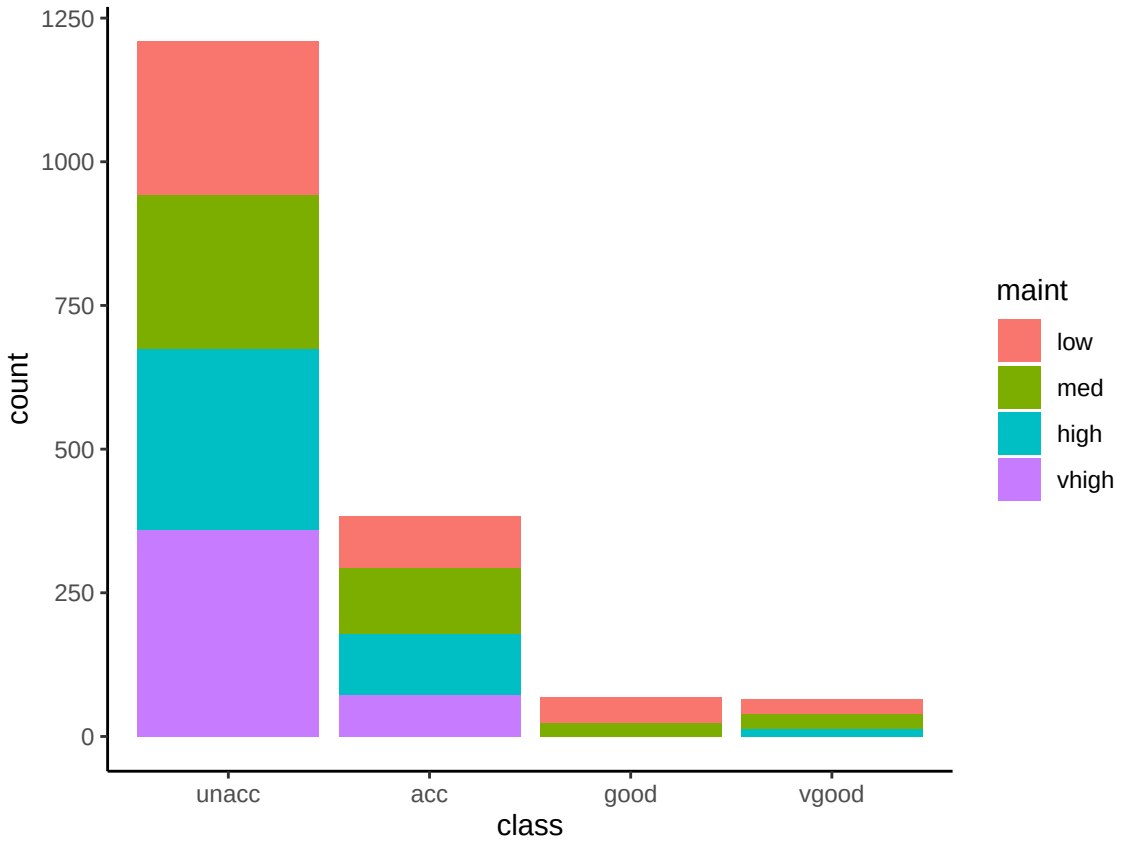
```
clean %>% ggplot(aes(x=class,fill=buying))+
            geom_histogram(stat="count")+
            theme_classic()
```



From the plot, we observe that the very good and good category vehicles have low or medium buying costs, while the acceptable and unacceptable category vehicles have low, medium, high and very high buying costs. This gives an idea that lower buying costs are correlated with higher category classification. This is expected since high cost vehicles will probably be less favoured by the buyers as compared to lower cost vehicles.

We can similarly plot the histogram with colours given by maintenance costs:

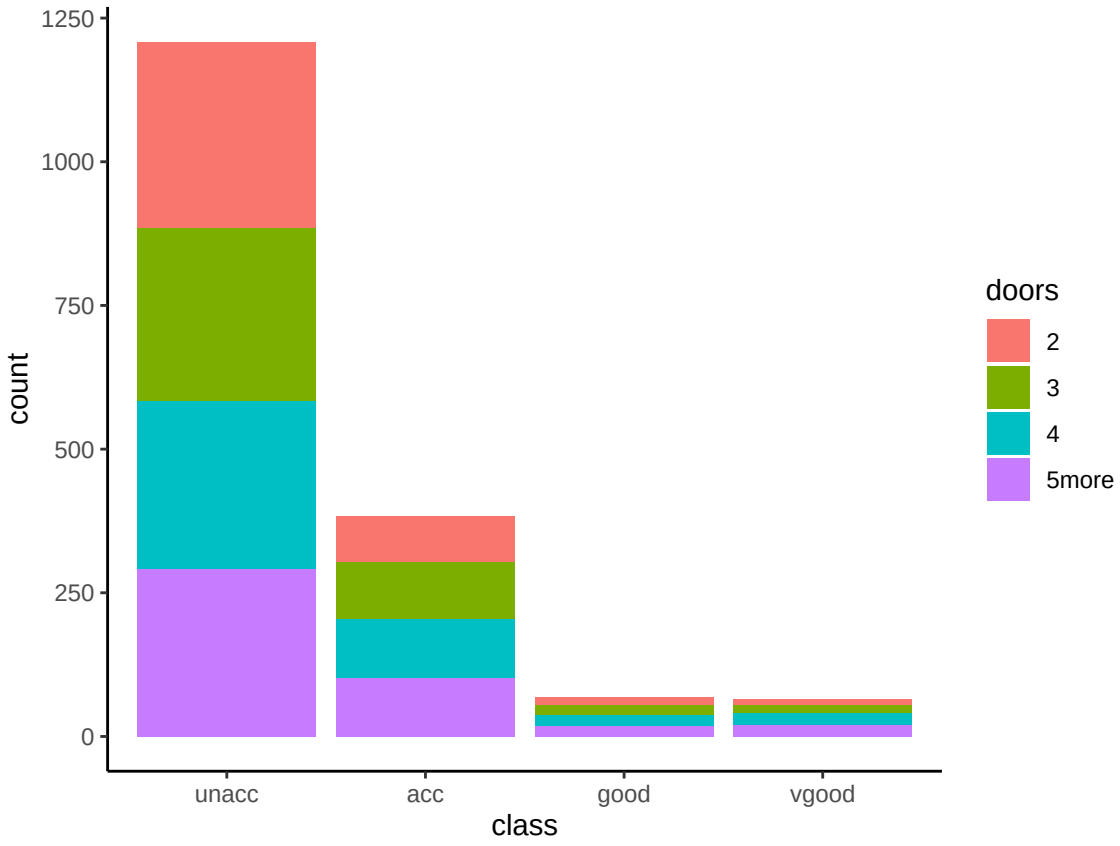
```
clean %>% ggplot(aes(x=class,fill=maint))+
            geom_histogram(stat="count")+
            theme_classic()
```



From the plot, we observe that very high and high maintenance costs are more correlated with unacceptable and acceptable vehicles, than with good and very good vehicles. This is expected since a prospective buyer will be less inclined to buy a vehicle with very high maintenance costs.

When we plot a histogram showing the number of doors in the four categories of vehicles:

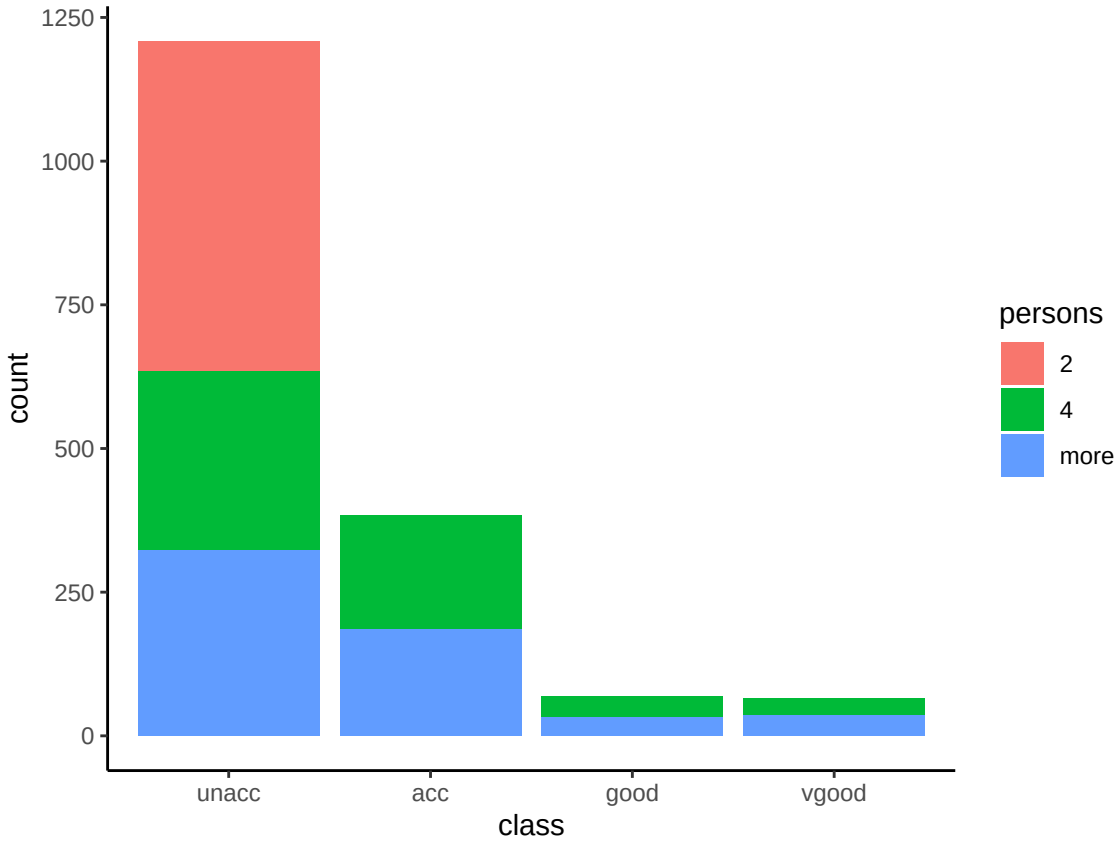
```
clean %>% ggplot(aes(x=class,fill=doors))+
            geom_histogram(stat="count")+
            theme_classic()
```



we observe that all four categories have vehicles with 2, 3, 4, 5 and more doors. This tells us that the number of doors is largely immaterial to the categorisation of vehicles.

Similarly, when we plot a histogram showing the number of persons that can sit in various categories of vehicles:

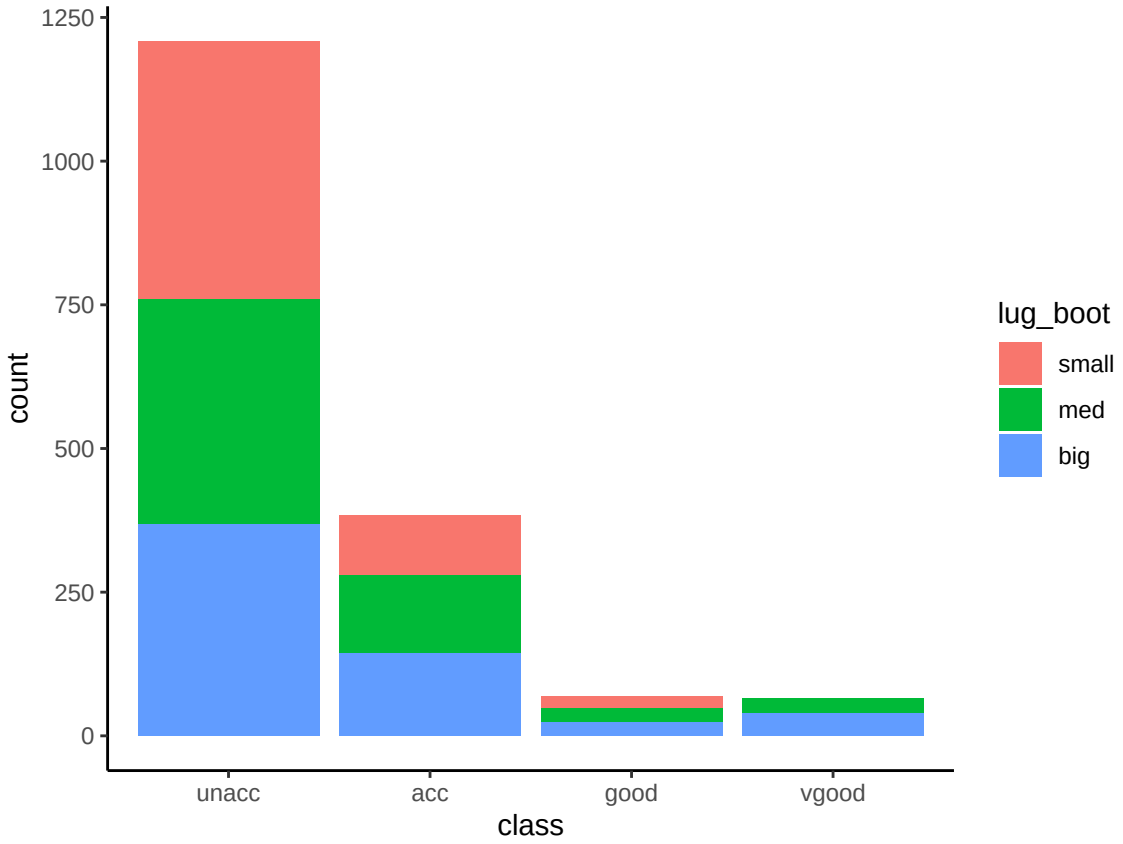
```
clean %>% ggplot(aes(x=class,fill=persons))+
            geom_histogram(stat="count")+
            theme_classic()
```



we observe that very good, good and acceptable vehicles sit 4 or more persons, while unacceptable vehicles sit 2, 4 or more vehicles. This indicates that a higher seating capacity is more correlated with higher categorisation of the vehicles. In particular, we note that all the vehicles with a seating capacity of 2 are unacceptable vehicles.

Plotting a histogram showing the luggage boot capacity of various categories of vehicles:

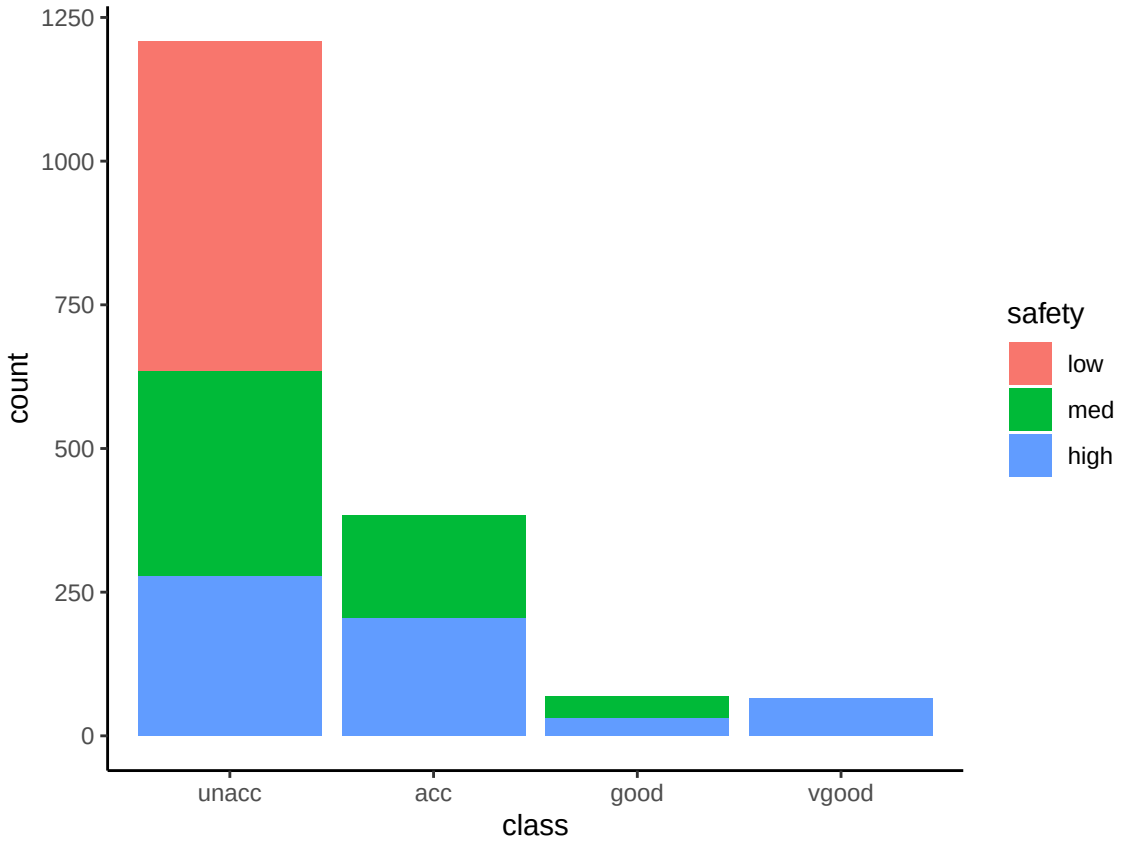
```
clean %>% ggplot(aes(x=class,fill=lug_boot))+
            geom_histogram(stat="count")+
            theme_classic()
```



shows that very good vehicles have medium or big-sized luggage boots, whereas other categories of vehicles have small, medium or big-sized luggage boots. This indicates that a larger-sized luggage boot is more preferable for higher categorisation of vehicles.

Plotting a histogram showing the safety classes of different categories of vehicles:

```
clean %>% ggplot(aes(x=class,fill=safety))+
            geom_histogram(stat="count")+
            theme_classic()
```



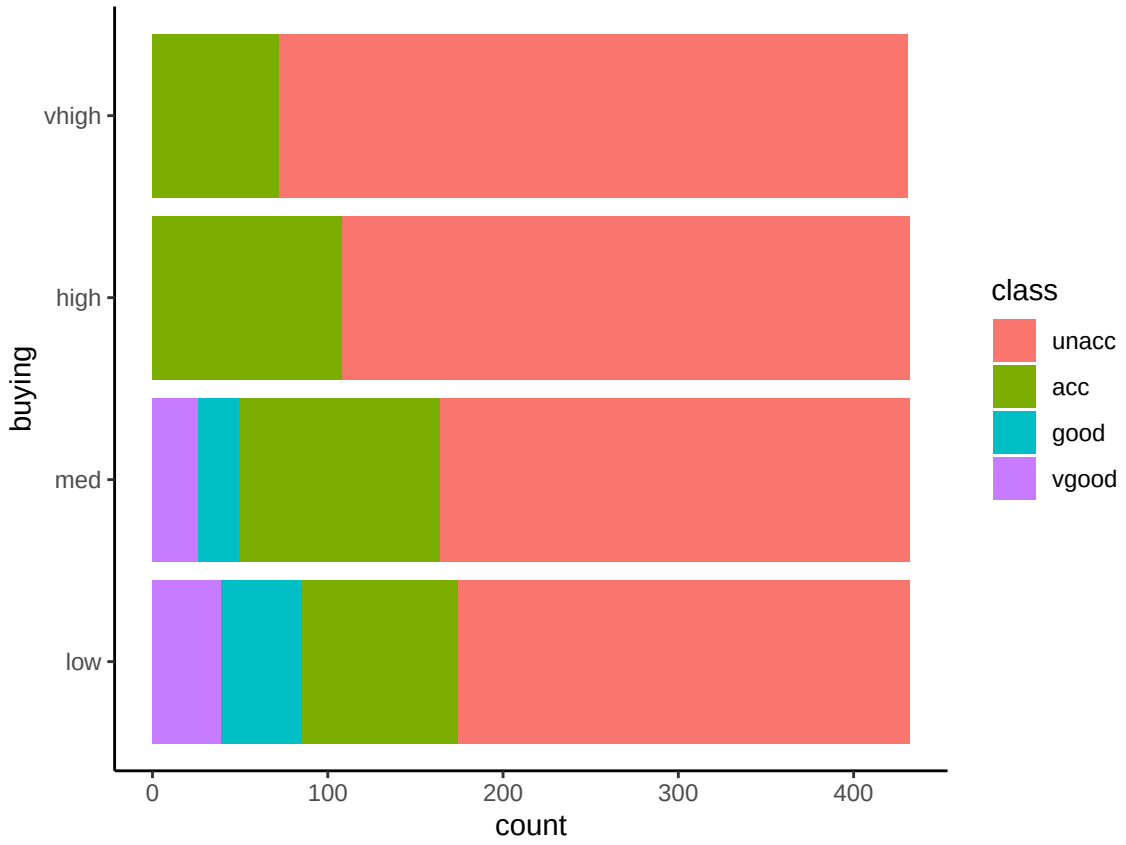
reveals that very good vehicles have high safety. Good and acceptable vehicles have high and medium safety ratings. Unacceptable vehicles have high, medium or low safety ratings. This indicates that a higher safety rating is more preferable for higher categorisation of vehicles. In particular, we also note that all the vehicles with a low safety rating are unacceptable vehicles.

Another way to visualise the data is by plotting the number of vehicles in different attributes. We represent the vehicle class in each bar through a different fill colour.

The attribute of buying cost:

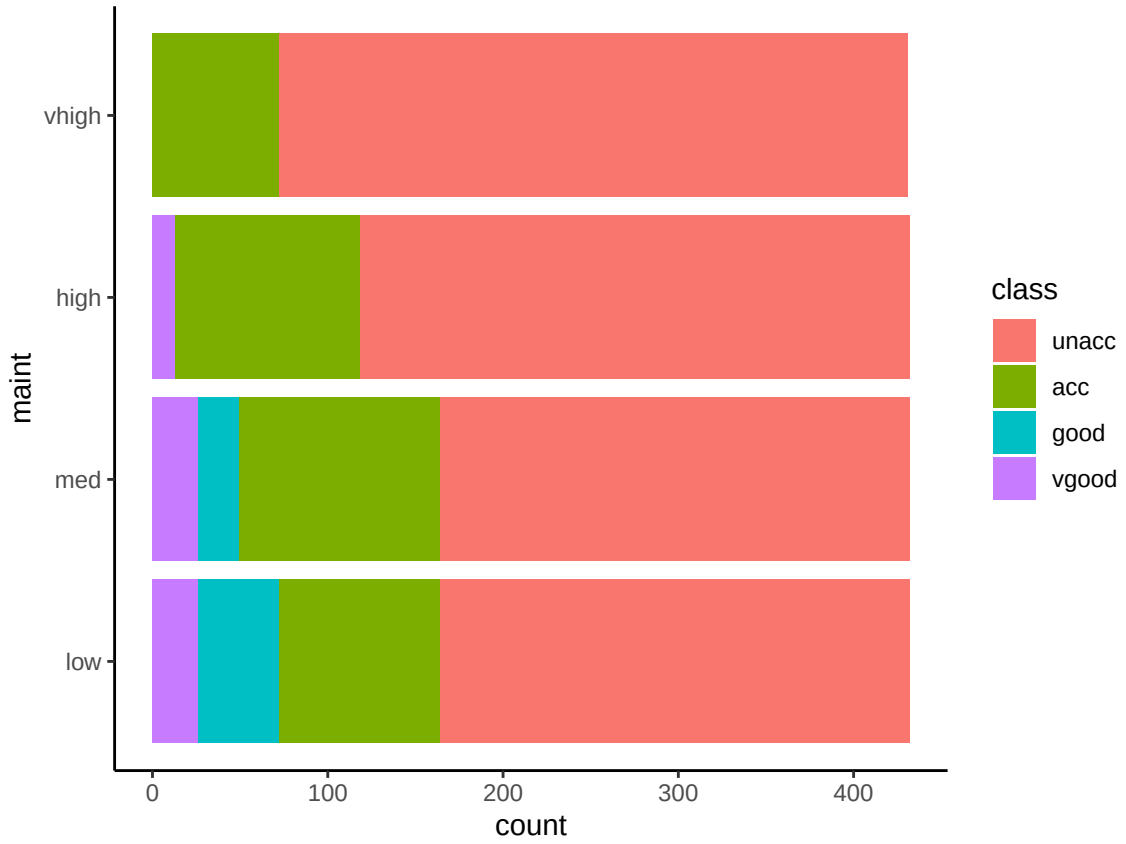
```
### Count vs. attribute, filled with class
clean %>% ggplot(aes(x=buying,fill=class))+
            geom_histogram(stat="count")+
```

```
            coord_flip()+theme_classic()
```



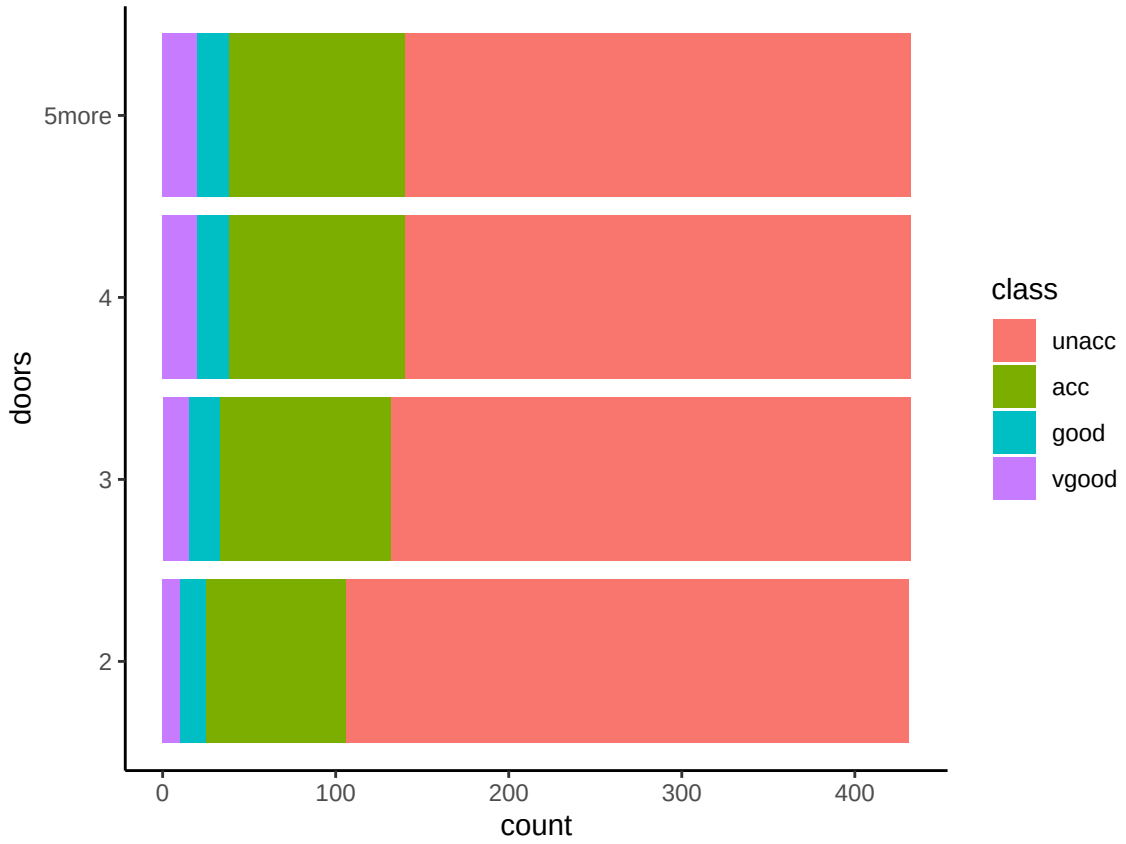
reveals that we have equal number of vehicles in each buying attribute class. This shows that the data is balanced. We find that all buying attribute classes have unacceptable and acceptable vehicles, while only low and medium buying attribute classes have good and very good vehicles. This indicates that lower buying costs are correlated with higher categorisation of vehicles. Similarly, the attribute of maintenance cost:

```
clean %>% ggplot(aes(x=maint,fill=class))+
            geom_histogram(stat="count")+
            coord_flip()+theme_classic()
```



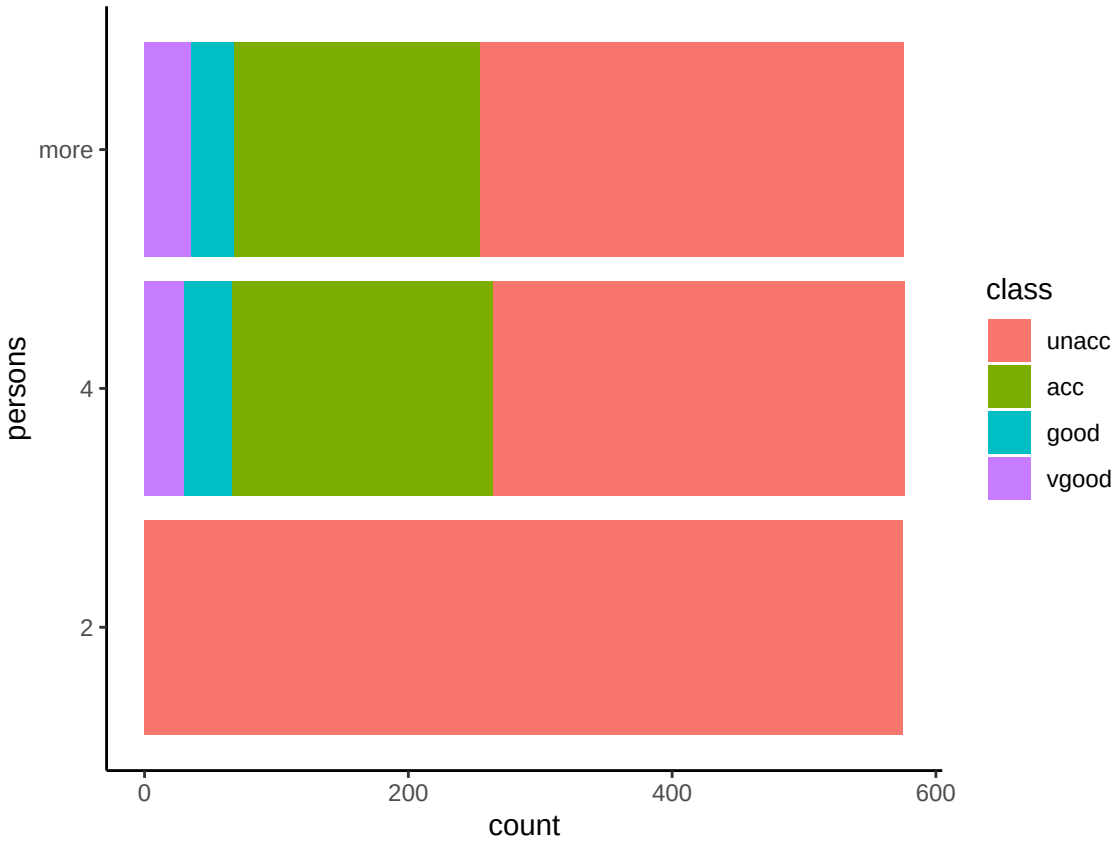
reveals that we have equal number of vehicles in each maintenance attribute class. This shows that the data is balanced. We find that all maintenance attribute classes have unacceptable and acceptable vehicles, while only low, medium and high maintenance attribute classes have good and very good vehicles. This indicates that lower maintenance costs are correlated with higher categorisation of vehicles. Similarly, the attribute of number of doors:

```
clean %>% ggplot(aes(x=doors,fill=class))+
            geom_histogram(stat="count")+
            coord_flip()+theme_classic()
```



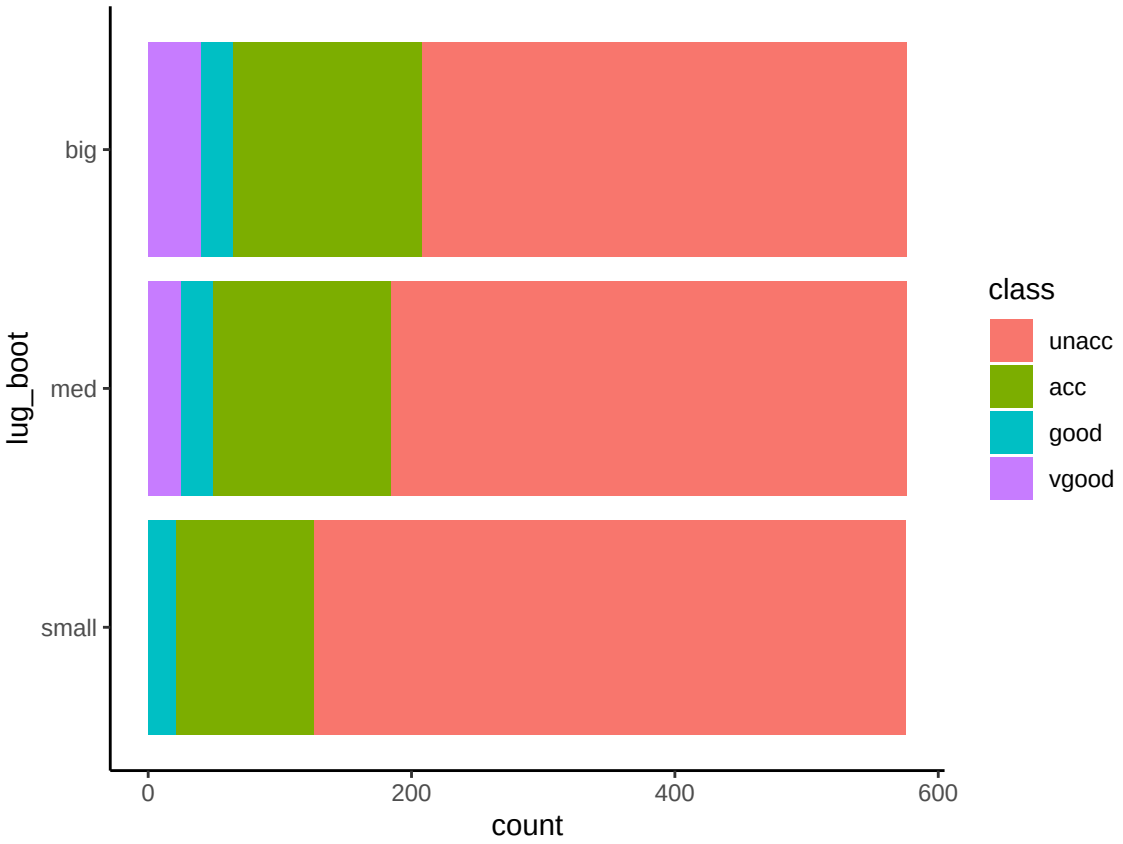
reveals that we have equal number of vehicles in each door attribute class. This shows that the data is balanced. We find that all door attribute classes have all classes of vehicles, indicating that the number of doors does not quite influence the categorisation of vehicles. Similarly, the attribute of persons:

```
clean %>% ggplot(aes(x=persons,fill=class))+
            geom_histogram(stat="count")+
            coord_flip()+theme_classic()
```



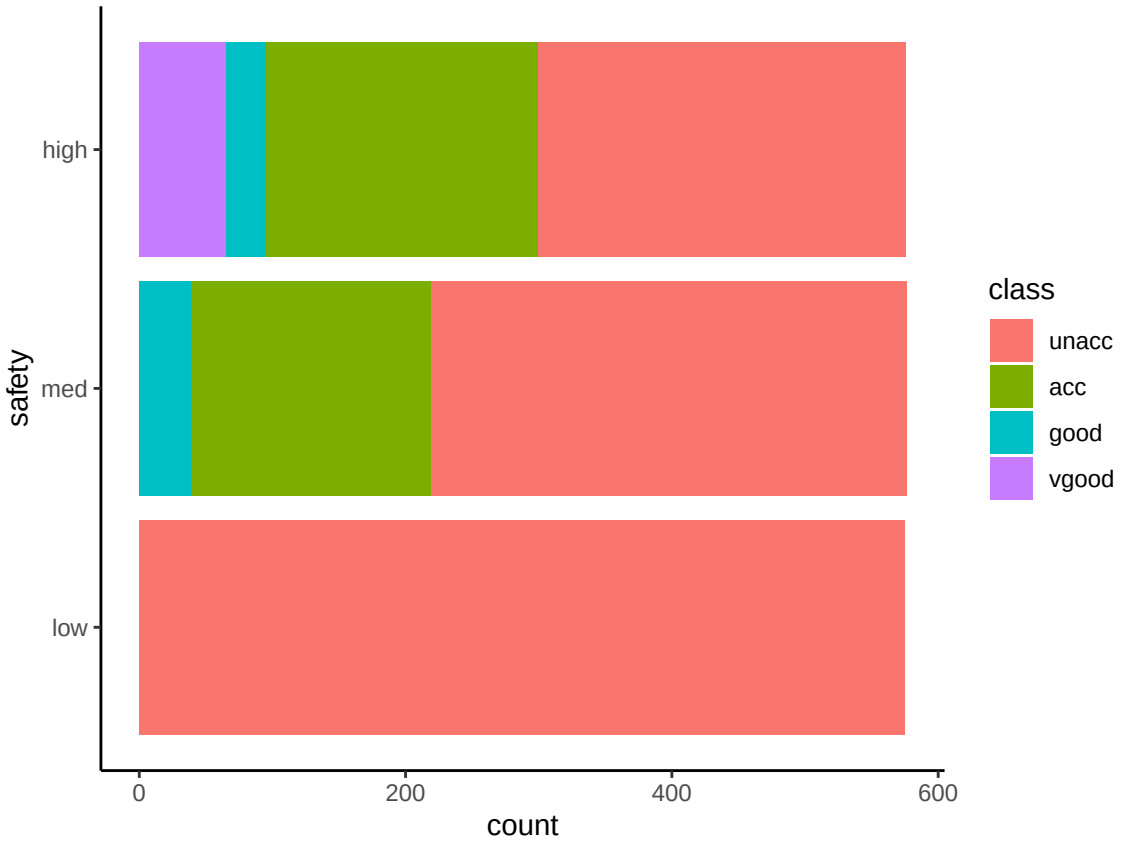
reveals that we have equal number of vehicles in each person attribute class. This shows that the data is balanced. We find that four and more person attribute classes have all categories of vehicles, while two person attribute class has only unacceptable vehicles. This indicates that lower person seating capacity is not favoured, and results in a lower categorisation of the vehicle. Similarly, the attribute of luggage boot capacity:

```
clean %>% ggplot(aes(x=lug_boot,fill=class))+
            geom_histogram(stat="count")+
            coord_flip()+theme_classic()
```



reveals that we have equal number of vehicles in each luggage boot attribute class. This shows that the data is balanced. We find that all luggage boot attribute classes have unacceptable, acceptable and good vehicles, while only medium and high luggage boot attribute classes have very good vehicles. This indicates that higher luggage boot capacity is preferred, and is correlated with higher categorisation of vehicles. Similarly, the attribute of safety:

```
clean %>% ggplot(aes(x=safety,fill=class))+
            geom_histogram(stat="count")+
            coord_flip()+theme_classic()
```



reveals that we have equal number of vehicles in each safety attribute class. This shows that the data is balanced. We find that low safety attribute class has only unacceptable vehicles, medium safety attribute class has unacceptable, acceptable, and good vehicles, and high safety attribute class has very good, good, acceptable and unacceptable vehicles. This indicates that higher safety attribute is correlated with higher categorisation of vehicles.

Insights gained

We have observed that while the data is unbalanced from the point of view of number of vehicles in different categories, it is quite balanced from the point of view of number of vehicles in different attribute categories. From the graphs, we observe that some attributes (like safety, seating capacity - persons, buying costs, and maintenance

costs) are strongly correlated to vehicle classification categories, while some other (such as the number of doors) are very weakly, if at all, correlated. This indicates that algorithms can be developed and trained to categorise vehicles in different classes.

Approach to modelling

Since there are several attributes that vary with vehicle classification, but there is no one-to-one rule evident from the correlations, we will prefer machine learning algorithmic prediction over causality. Different algorithms will be trained and tested to select the one with the best accuracy. The selected one(s) will be validated using the validation dataset.

Results and discussion

Dividing the dataset into working and validation datasets

We use createDataPartition() function from the caret package to partition the dataset into working and validation datasets. We set the seed to 1 to ensure replicability of the submitted results. As explained in the key steps section, we partition the dataset in 90:10 manner, with the validation dataset being 10% and the working dataset being 90%.

```
### Dividing the dataset into working and validation datasets.
# Validation dataset will not be used till the end.
set.seed(1)
test_index <- createDataPartition(y = clean$buying,
                times = 1, p = 0.1, list = FALSE)
working <- clean[-test_index,]
validation <- clean[test_index,]
```

The createDataPartition() function outputs 10% indices randomly; these are stored in the test_index variable. clean[test_index,] outputs the rows with the selected indices; these are stored as the validation dataset. clean[-test_index,] outputs the rows without the selected indices; these are stored as the working dataset. We check the sizes of these two partitions using the dim() function:

```
### Check dimensions of working and validation datasets to
# ensure that the partitioning has been done as intended.
dim(working)
```

```
## [1] 1551    7
```

```
dim(validation)
```

```
## [1] 176   7
```

The working dataset has 1551 rows and 7 columns, while the validation dataset has 176 rows and 7 columns. This is approximately a 90:10 partition, as intended.

We next use createDataPartition() function from the caret package to partition the working dataset into test and train datasets. We set the seed to 1 to ensure replicability of the submitted results. As explained in the key steps section, we partition the dataset in 90:10 manner, with the test dataset being 10% and the train dataset being 90%.

Dividing working dataset into train and test datasets

```
### Dividing the working dataset into train and test datasets.
# The train dataset is used to train the algorithms, and the
# test dataset is used to test the algorithms and choose the
# one with the highest accuracy. The selected algorithm(s)
# will be validated on the validation dataset at the end.
set.seed(1)
test_index <- createDataPartition(y = working$buying,
                times = 1, p = 0.1, list = FALSE)
train <- working[-test_index,]
test <- working[test_index,]
```

The createDataPartition() function outputs 10% indices randomly; these are stored in the test_index variable. working[test_index,] outputs the rows with the selected indices; these are stored as the test dataset. working[-test_index,] outputs the rows without the selected indices; these are stored as the train dataset. We check the sizes of these two partitions using the dim() function:

```
### Check dimensions of train and test datasets to
# ensure that the partitioning has been done as
# intended.
dim(train)
```

```
## [1] 1395    7
```

```
dim(test)
```

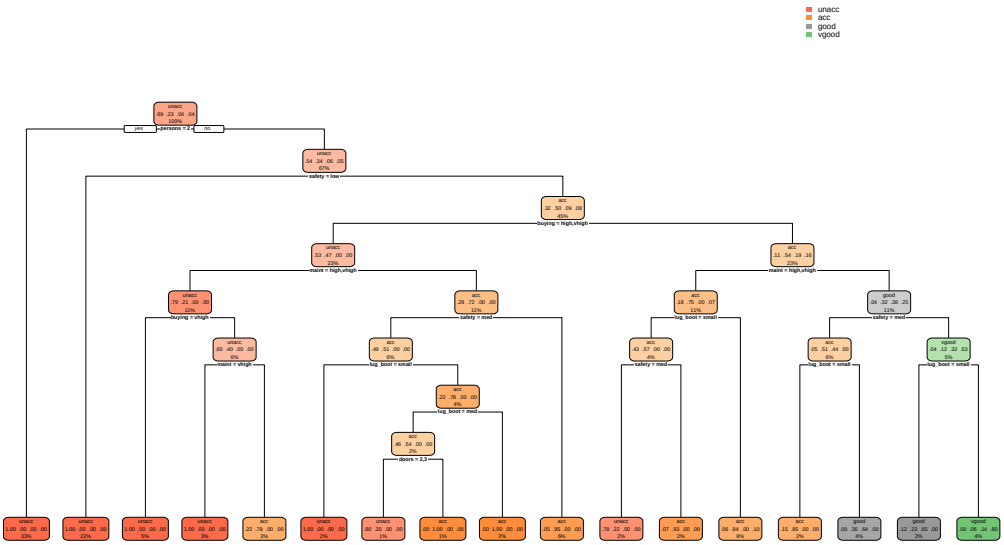
```
## [1] 156   7
```

The train dataset has 1395 rows and 7 columns, while the test dataset has 156 rows and 7 columns. This is approximately a 90:10 partition, as intended.

Making decision tree

The decision tree helps classify the dataset as a human would do it, by making a flowchart that can be followed to classify cars into different categories based on their characteristics. Here we make the decision tree using the rpart method. We make two decision trees, one using the caret package, and the other using the rpart function. We also make two kinds of decision trees using prp and rpart.plot functions.

```
################## Making decision tree - 1 ##################

set.seed(1)

fit<-rpart(formula = class ~ .,data=train,method = "class")

rpart.plot(fit)
```



We observe that in this decision tree, we begin with a car defaultly categorised as an unacceptable car. The first node asks about the number of persons. If the car seats 2 persons, it is finally categorised as an unacceptable car. If no, the next node asks for the safety rating of the vehicle. If the safety rating is low, the car is finally categorised as unacceptable. If the safety rating is not low, we reach the node asking if the buying price is high or very high. And so on.

At each stage, we also get the proportion of vehicles categorised into particular classes. By following the flowchart, we can classify any vehicle into the set categories.

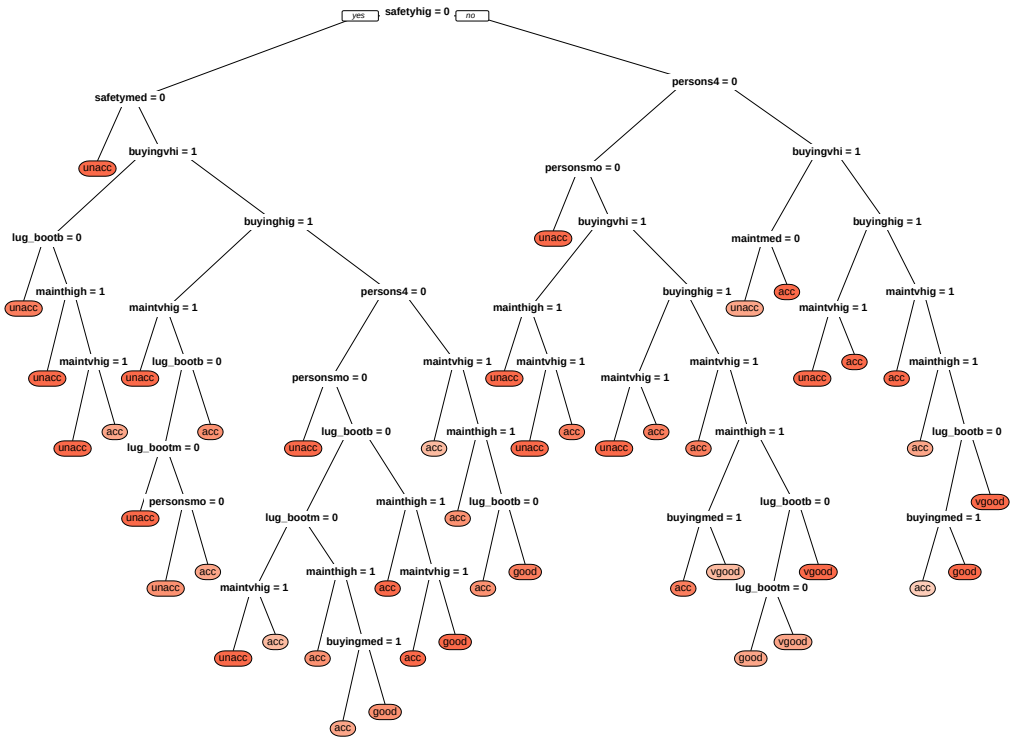
We also make a decision tree using the train() function in caret package.

```r
################## Making decision tree - 2 ##################
set.seed(1)
dtree_fit <- train(class ~., data = train, method = "rpart",
                   parms = list(split = "information"),
                   tuneLength = 10)
dtree_fit
```

```
## CART
##
## 1395 samples
##    6 predictor
##    4 classes: 'unacc', 'acc', 'good', 'vgood'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1395, 1395, 1395, 1395, 1395, 1395, ...
## Resampling results across tuning parameters:
##
##   cp        Accuracy  Kappa
##   0.001554  0.8633    0.7103
##   0.002331  0.8622    0.7076
##   0.004662  0.8599    0.7018
##   0.008159  0.8586    0.6971
##   0.011655  0.8458    0.6670
##   0.013986  0.8302    0.6312
```

```
##    0.014375  0.8252    0.6190
##    0.015152  0.8180    0.6032
##    0.022533  0.7826    0.5128
##    0.066434  0.7344    0.2583
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.001554.
```

```
prp(dtree_fit$finalModel, box.palette = "Reds", tweak = 1.2)
```



We observe that the algorithm considers several complexity parameters (cp) to choose the most optimal model. In this case, the optimal model was arrived with cp = 0.001554. This chosen model gives an accuracy of 86.33%. We also get the flowchart with several bifurcations.

We also test the chosen model against the test dataset. To do this, we make use of the

predict() function to predict the classification values using the test dataset attributes and construct a confusion matrix:

```
# Inspecting the confusion matrix
pred <- predict(dtree_fit, newdata = test)
confusionMatrix(pred, as.factor(test$class))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction unacc acc good vgood
##      unacc   108   5    0     0
##      acc       6  23    1     1
##      good      0   2    3     1
##      vgood     0   0    0     6
##
## Overall Statistics
##
##                Accuracy : 0.897
##                  95% CI : (0.839, 0.94)
##     No Information Rate : 0.731
##     P-Value [Acc > NIR] : 2.45e-07
##
##                   Kappa : 0.761
##
##  Mcnemar's Test P-Value : NA
```

```
##
## Statistics by Class:
##
##                      Class: unacc Class: acc Class: good Class: vgood
## Sensitivity                 0.947      0.767      0.7500       0.7500
## Specificity                 0.881      0.937      0.9803       1.0000
## Pos Pred Value              0.956      0.742      0.5000       1.0000
## Neg Pred Value              0.860      0.944      0.9933       0.9867
## Prevalence                  0.731      0.192      0.0256       0.0513
## Detection Rate              0.692      0.147      0.0192       0.0385
## Detection Prevalence        0.724      0.199      0.0385       0.0385
## Balanced Accuracy           0.914      0.852      0.8651       0.8750
```

The confusion matrix reveals that most of the vehicles are classified correctly. For example, 108 unacceptable vehicles were classified as unacceptable, 23 acceptable vehicles were classified as acceptable, 3 good vehicles were classified as good, and 6 very good vehicles were classifed as very good. For the vehicles that were incorrectly classified, we observe that their classification was in the nearest categories. For example, 6 unacceptable vehicles were classified as acceptable, but no unacceptable vehicle was classified into the good or very good category. Similarly, of the 7 acceptable vehicles wrongly classified, 5 were classified to be unacceptable, and 2 were classified as good. No acceptable vehicle was wrongly classified as a very good vehicle. Only 1 good vehicle was wrongly classified as an acceptable vehicle. 1 very good vehicle was wrongly classified as good vehicle, and 1 very good vehicle was wrongly classified as an acceptable vehicle.

These observations reveal that the method is quite robust with good accuracy (of 89.7%) and no large errors in vehicle classification. We also obtain the class-wise sensitivity, specificity, balanced accuracy, and other parameters for this algorithm.

Let us now explore if we can improve the results by using other algorithms.

Training and testing algorithms

We train various machine learning algorithms on the train dataset using the train() function from caret package. The trained algorithm is stored in the fit variable, which is then used to predict the car classes in the test dataset. The confusion matrix is computed between the predicted classification and the actual classification using the confusionMatrix() function. The confusion matrix is then analysed to get an idea of how good the trained algorithm performs on the test dataset. We also peruse the overall statistics, particularly the accuracy. We look at class-wise statistics including sensitivity, specificity, positive prediction value and the balanced accuracy. Then a table is created to record the accuracy and the F1 values for different classes.

The first method is the knn, or the k-nearest neighbours:

```
### Training and testing various algorithms ###
################### knn ###################
set.seed(1)
fit <- train(class ~ ., method = "knn", data = train)
pred <- predict(fit, newdata = test)
cM <- confusionMatrix(pred, as.factor(test$class))
cM
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction unacc acc good vgood
##      unacc   113  13    1     1
##      acc       1  16    1     1
##      good      0   1    2     0
##      vgood     0   0    0     6
##
## Overall Statistics
##
##                Accuracy : 0.878
##                  95% CI : (0.816, 0.925)
##     No Information Rate : 0.731
##     P-Value [Acc > NIR] : 6.22e-06
##
##                   Kappa : 0.675
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: unacc Class: acc Class: good Class: vgood
## Sensitivity                 0.991      0.533      0.5000       0.7500
## Specificity                 0.643      0.976      0.9934       1.0000
## Pos Pred Value              0.883      0.842      0.6667       1.0000
```

```
## Neg Pred Value              0.964    0.898    0.9869       0.9867
## Prevalence                  0.731    0.192    0.0256       0.0513
## Detection Rate              0.724    0.103    0.0128       0.0385
## Detection Prevalence        0.821    0.122    0.0192       0.0385
## Balanced Accuracy           0.817    0.755    0.7467       0.8750
```

```
acc <- cM$overall['Accuracy']
F1unacc <- cM$byClass[1,'F1']
F1acc <- cM$byClass[2,'F1']
F1good <- cM$byClass[3,'F1']
F1vgood <- cM$byClass[4,'F1']

accuracy <- data_frame(method = "knn", accuracy = acc,
                       F1unacc = F1unacc, F1acc = F1acc,
                       F1good = F1good, F1vgood = F1vgood)

accuracy %>% knitr::kable()
```

method	accuracy	F1unacc	F1acc	F1good	F1vgood
knn	0.8782	0.9339	0.6531	0.5714	0.8571

This method computes the distances between the query and all the examples in the data. It then selects a specified number examples (K) closest to the query, voting for the most frequent label, which is then used for classification.

We find that the knn method gives an overall accuracy of 87.82%, with the F1 values

of unacceptable, acceptable, good and very good classes as 0.9339, 0.6531, 0.5714, and 0.8571, respectively.

Next we train the lda, or the linear dimensionality algorithm:

```
################### lda ###################
set.seed(1)
fit <- train(class ~ ., method = "lda", data = train)
pred <- predict(fit, newdata = test)
cM <- confusionMatrix(pred, as.factor(test$class))
cM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction unacc acc good vgood
##      unacc   109   4    0     0
##      acc       5  25    3     2
##      good      0   1    1     1
##      vgood     0   0    0     5
##
## Overall Statistics
##
##                Accuracy : 0.897
##                  95% CI : (0.839, 0.94)
##     No Information Rate : 0.731
##     P-Value [Acc > NIR] : 2.45e-07
```

```
##
##                  Kappa : 0.759
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: unacc Class: acc Class: good Class: vgood
## Sensitivity                 0.956      0.833     0.25000        0.6250
## Specificity                 0.905      0.921     0.98684        1.0000
## Pos Pred Value              0.965      0.714     0.33333        1.0000
## Neg Pred Value              0.884      0.959     0.98039        0.9801
## Prevalence                  0.731      0.192     0.02564        0.0513
## Detection Rate              0.699      0.160     0.00641        0.0321
## Detection Prevalence        0.724      0.224     0.01923        0.0321
## Balanced Accuracy           0.930      0.877     0.61842        0.8125
```

```
acc <- cM$overall['Accuracy']
F1unacc <- cM$byClass[1,'F1']
F1acc <- cM$byClass[2,'F1']
F1good <- cM$byClass[3,'F1']
F1vgood <- cM$byClass[4,'F1']


accuracy <- bind_rows(accuracy,
                      data_frame(method="lda",
```

```
                            accuracy = acc,
                            F1unacc = F1unacc, F1acc = F1acc,
                            F1good = F1good, F1vgood = F1vgood))
accuracy %>% knitr::kable()
```

method	accuracy	F1unacc	F1acc	F1good	F1vgood
knn	0.8782	0.9339	0.6531	0.5714	0.8571
lda	0.8974	0.9604	0.7692	0.2857	0.7692

This method reduces the number of variables in the dataset while retaining as much information as possible. This information is then used for classification.

We find that the lda method gives an overall accuracy of 89.74%, with the F1 values of unacceptable, acceptable, good and very good classes as 0.9604, 0.7692, 0.2857, and 0.7692, respectively.

Next we train the naive_bayes, or the naive bayes algorithm:

```
################### naive_bayes ###################
set.seed(1)
fit <- train(class ~ ., method = "naive_bayes", data = train)
pred <- predict(fit, newdata = test)
cM <- confusionMatrix(pred, as.factor(test$class))
cM
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction unacc acc good vgood
##      unacc   114  30    4     8
##      acc       0   0    0     0
##      good      0   0    0     0
##      vgood     0   0    0     0
##
## Overall Statistics
##
##                Accuracy : 0.731
##                  95% CI : (0.654, 0.799)
##     No Information Rate : 0.731
##     P-Value [Acc > NIR] : 0.541
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: unacc Class: acc Class: good Class: vgood
## Sensitivity                 1.000      0.000      0.0000       0.0000
## Specificity                 0.000      1.000      1.0000       1.0000
## Pos Pred Value              0.731        NaN         NaN          NaN
## Neg Pred Value                NaN      0.808      0.9744       0.9487
```

```
## Prevalence                 0.731      0.192      0.0256       0.0513
## Detection Rate             0.731      0.000      0.0000       0.0000
## Detection Prevalence       1.000      0.000      0.0000       0.0000
## Balanced Accuracy          0.500      0.500      0.5000       0.5000
```

```
acc <- cM$overall['Accuracy']
F1unacc <- cM$byClass[1,'F1']
F1acc <- cM$byClass[2,'F1']
F1good <- cM$byClass[3,'F1']
F1vgood <- cM$byClass[4,'F1']

accuracy <- bind_rows(accuracy,
                      data_frame(method="naive_bayes",
                                 accuracy = acc,
                                 F1unacc = F1unacc, F1acc = F1acc,
                                 F1good = F1good, F1vgood = F1vgood))
accuracy %>% knitr::kable()
```

method	accuracy	F1unacc	F1acc	F1good	F1vgood
knn	0.8782	0.9339	0.6531	0.5714	0.8571
lda	0.8974	0.9604	0.7692	0.2857	0.7692
naive_bayes	0.7308	0.8444	NA	NA	NA

This method uses a supervised learning algorithm that applies Bayes' theorem with the "naive" assumption of conditional independence between every pair of features.

Bayes' theorem is used for classification.

We find that the naive_bayes method gives an overall accuracy of 73.08%, with the F1 values of unacceptable, acceptable, good and very good classes as 0.8444, NA, NA, and NA, respectively. This is because this algorithm predicts all vehicles to belong to the unacceptable class. In this way, this algorithm is unsuitable for our classification exercise.

Next we train the svmLinear, or the support vector machines algorithm:

```
################### svmLinear ###################
set.seed(1)
fit <- train(class ~ ., method = "svmLinear", data = train)
pred <- predict(fit, newdata = test)
cM <- confusionMatrix(pred, as.factor(test$class))
cM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction unacc acc good vgood
##      unacc   110   4    0     0
##      acc       4  25    0     0
##      good      0   1    4     0
##      vgood     0   0    0     8
##
## Overall Statistics
##
```

```
##                 Accuracy : 0.942
##                   95% CI : (0.893, 0.973)
##     No Information Rate : 0.731
##     P-Value [Acc > NIR] : 9.99e-12
##
##                   Kappa : 0.865
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: unacc Class: acc Class: good Class: vgood
## Sensitivity                 0.965      0.833      1.0000       1.0000
## Specificity                 0.905      0.968      0.9934       1.0000
## Pos Pred Value              0.965      0.862      0.8000       1.0000
## Neg Pred Value              0.905      0.961      1.0000       1.0000
## Prevalence                  0.731      0.192      0.0256       0.0513
## Detection Rate              0.705      0.160      0.0256       0.0513
## Detection Prevalence        0.731      0.186      0.0321       0.0513
## Balanced Accuracy           0.935      0.901      0.9967       1.0000
```

```r
acc <- cM$overall['Accuracy']
F1unacc <- cM$byClass[1,'F1']
F1acc <- cM$byClass[2,'F1']
F1good <- cM$byClass[3,'F1']
```

```
F1vgood <- cM$byClass[4,'F1']


accuracy <- bind_rows(accuracy,
                      data_frame(method="svmLinear",
                                 accuracy = acc,
                                 F1unacc = F1unacc, F1acc = F1acc,
                                 F1good = F1good, F1vgood = F1vgood))
accuracy %>% knitr::kable()
```

method	accuracy	F1unacc	F1acc	F1good	F1vgood
knn	0.8782	0.9339	0.6531	0.5714	0.8571
lda	0.8974	0.9604	0.7692	0.2857	0.7692
naive_bayes	0.7308	0.8444	NA	NA	NA
svmLinear	0.9423	0.9649	0.8475	0.8889	1.0000

This method constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which is then used for classification.

We find that the svmLinear method gives an overall accuracy of 94.23%, with the F1 values of unacceptable, acceptable, good and very good classes as 0.9649, 0.8475, 0.8889, and 1.0000, respectively.

Next we train the treebag algorithm:

```
################### treebag ###################
set.seed(1)
fit <- train(class ~ ., method = "treebag", data = train)
```

```r
pred <- predict(fit, newdata = test)
cM <- confusionMatrix(pred, as.factor(test$class))
cM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction unacc acc good vgood
##      unacc   113   8    0     0
##      acc       1  21    0     1
##      good      0   1    4     1
##      vgood     0   0    0     6
##
## Overall Statistics
##
##                Accuracy : 0.923
##                  95% CI : (0.869, 0.96)
##     No Information Rate : 0.731
##     P-Value [Acc > NIR] : 1.26e-09
##
##                   Kappa : 0.809
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
```

```
##
##                      Class: unacc Class: acc Class: good Class: vgood
## Sensitivity                 0.991      0.700      1.0000       0.7500
## Specificity                 0.810      0.984      0.9868       1.0000
## Pos Pred Value              0.934      0.913      0.6667       1.0000
## Neg Pred Value              0.971      0.932      1.0000       0.9867
## Prevalence                  0.731      0.192      0.0256       0.0513
## Detection Rate              0.724      0.135      0.0256       0.0385
## Detection Prevalence        0.776      0.147      0.0385       0.0385
## Balanced Accuracy           0.900      0.842      0.9934       0.8750
```

```r
acc <- cM$overall['Accuracy']
F1unacc <- cM$byClass[1,'F1']
F1acc <- cM$byClass[2,'F1']
F1good <- cM$byClass[3,'F1']
F1vgood <- cM$byClass[4,'F1']


accuracy <- bind_rows(accuracy,
                      data_frame(method="treebag",
                                 accuracy = acc,
                                 F1unacc = F1unacc, F1acc = F1acc,
                                 F1good = F1good, F1vgood = F1vgood))
accuracy %>% knitr::kable()
```

method	accuracy	F1unacc	F1acc	F1good	F1vgood
knn	0.8782	0.9339	0.6531	0.5714	0.8571
lda	0.8974	0.9604	0.7692	0.2857	0.7692
naive_bayes	0.7308	0.8444	NA	NA	NA
svmLinear	0.9423	0.9649	0.8475	0.8889	1.0000
treebag	0.9231	0.9617	0.7925	0.8000	0.8571

This method combines the results of many decision trees, reducing the effects of overfitting and improving generalization, which is then used for classification.

We find that the treebag method gives an overall accuracy of 92.31%, with the F1 values of unacceptable, acceptable, good and very good classes as 0.9617, 0.7925, 0.8000, and 0.8571, respectively.

Next we train the gamLoess, or the generalised additive model using LOESS:

```
################### gamLoess ###################
set.seed(1)
fit <- train(class ~ ., method = "gamLoess", data = train)
pred <- predict(fit, newdata = test)
cM <- confusionMatrix(pred, as.factor(test$class))
cM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction unacc acc good vgood
```

```
##      unacc   110   5    0     0
##      acc       4  25    4     8
##      good      0   0    0     0
##      vgood     0   0    0     0
##
## Overall Statistics
##
##                Accuracy : 0.865
##                  95% CI : (0.802, 0.915)
##     No Information Rate : 0.731
##     P-Value [Acc > NIR] : 4.01e-05
##
##                   Kappa : 0.672
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: unacc Class: acc Class: good Class: vgood
## Sensitivity                 0.965      0.833      0.0000       0.0000
## Specificity                 0.881      0.873      1.0000       1.0000
## Pos Pred Value              0.957      0.610         NaN          NaN
## Neg Pred Value              0.902      0.957      0.9744       0.9487
## Prevalence                  0.731      0.192      0.0256       0.0513
## Detection Rate              0.705      0.160      0.0000       0.0000
```

```
## Detection Prevalence         0.737      0.263      0.0000       0.0000
## Balanced Accuracy            0.923      0.853      0.5000       0.5000
```

```
acc <- cM$overall['Accuracy']
F1unacc <- cM$byClass[1,'F1']
F1acc <- cM$byClass[2,'F1']
F1good <- cM$byClass[3,'F1']
F1vgood <- cM$byClass[4,'F1']

accuracy <- bind_rows(accuracy,
                      data_frame(method="gamLoess",
                                 accuracy = acc,
                                 F1unacc = F1unacc, F1acc = F1acc,
                                 F1good = F1good, F1vgood = F1vgood))
accuracy %>% knitr::kable()
```

method	accuracy	F1unacc	F1acc	F1good	F1vgood
knn	0.8782	0.9339	0.6531	0.5714	0.8571
lda	0.8974	0.9604	0.7692	0.2857	0.7692
naive_bayes	0.7308	0.8444	NA	NA	NA
svmLinear	0.9423	0.9649	0.8475	0.8889	1.0000
treebag	0.9231	0.9617	0.7925	0.8000	0.8571
gamLoess	0.8654	0.9607	0.7042	NA	NA

This method considers that the target variable is the sum of a non-linear combina-

tion of variables, and combines this information with locally estimated scatterplot smoothing. The model is then used for classification.

We find that the gamLoess method gives an overall accuracy of 86.54%, with the F1 values of unacceptable, acceptable, good and very good classes as 0.9607, 0.7042, NA, and NA, respectively. This is because it does not classify any vehicle in the good or very good class, as evident by the confusion matrix. This method is, thus, unsuitable for our classification purpose.

Next we train the gam, or the generalised additive model:

```
################### gam ###################
set.seed(1)
fit <- train(class ~ ., method = "gam", data = train)
```

```
## Loading required package: mgcv
```

```
## Loading required package: nlme
```

```
##
## Attaching package: 'nlme'
```

```
## The following object is masked from 'package:dplyr':
##
##     collapse
```

```
## This is mgcv 1.8-35. For overview type 'help("mgcv-package")'.
```

```
##
## Attaching package: 'mgcv'
```

```
## The following objects are masked from 'package:gam':
##
##     gam, gam.control, gam.fit, s
```

```
pred <- predict(fit, newdata = test)
cM <- confusionMatrix(pred, as.factor(test$class))
cM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction unacc acc good vgood
##      unacc   110   5    0     0
##      acc       4  25    4     8
##      good      0   0    0     0
##      vgood     0   0    0     0
##
## Overall Statistics
##
##                Accuracy : 0.865
##                  95% CI : (0.802, 0.915)
##     No Information Rate : 0.731
##     P-Value [Acc > NIR] : 4.01e-05
##
##                   Kappa : 0.672
##
```

```
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: unacc Class: acc Class: good Class: vgood
## Sensitivity                 0.965      0.833      0.0000       0.0000
## Specificity                 0.881      0.873      1.0000       1.0000
## Pos Pred Value              0.957      0.610         NaN          NaN
## Neg Pred Value              0.902      0.957      0.9744       0.9487
## Prevalence                  0.731      0.192      0.0256       0.0513
## Detection Rate              0.705      0.160      0.0000       0.0000
## Detection Prevalence        0.737      0.263      0.0000       0.0000
## Balanced Accuracy           0.923      0.853      0.5000       0.5000
```

```
acc <- cM$overall['Accuracy']
F1unacc <- cM$byClass[1,'F1']
F1acc <- cM$byClass[2,'F1']
F1good <- cM$byClass[3,'F1']
F1vgood <- cM$byClass[4,'F1']

accuracy <- bind_rows(accuracy,
                      data_frame(method="gam",
                           accuracy = acc,
                           F1unacc = F1unacc, F1acc = F1acc,
                           F1good = F1good, F1vgood = F1vgood))
```

```
accuracy %>% knitr::kable()
```

method	accuracy	F1unacc	F1acc	F1good	F1vgood
knn	0.8782	0.9339	0.6531	0.5714	0.8571
lda	0.8974	0.9604	0.7692	0.2857	0.7692
naive_bayes	0.7308	0.8444	NA	NA	NA
svmLinear	0.9423	0.9649	0.8475	0.8889	1.0000
treebag	0.9231	0.9617	0.7925	0.8000	0.8571
gamLoess	0.8654	0.9607	0.7042	NA	NA
gam	0.8654	0.9607	0.7042	NA	NA

This method considers that the target variable is the sum of a non-linear combination of variables, and uses this information for classification.

We find that the gam method gives an overall accuracy of 86.54%, with the F1 values of unacceptable, acceptable, good and very good classes as 0.9607, 0.7042, NA, and NA, respectively. This is because it does not classify any vehicle in the good or very good class, as evident by the confusion matrix. This method is, thus, unsuitable for our classification purpose.

Next we train the svmRadial, or the support vector machine model with radial basis:

```
################### svmRadial ###################
set.seed(1)
fit <- train(class ~ ., method = "svmRadial", data = train)
pred <- predict(fit, newdata = test)
cM <- confusionMatrix(pred, as.factor(test$class))
```

```
cM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction unacc acc good vgood
##      unacc   112   1    0     0
##      acc       2  29    1     1
##      good      0   0    2     0
##      vgood     0   0    1     7
##
## Overall Statistics
##
##                Accuracy : 0.962
##                  95% CI : (0.918, 0.986)
##     No Information Rate : 0.731
##     P-Value [Acc > NIR] : 2.85e-14
##
##                   Kappa : 0.91
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: unacc Class: acc Class: good Class: vgood
```

```
## Sensitivity                 0.982      0.967      0.5000       0.8750
## Specificity                 0.976      0.968      1.0000       0.9932
## Pos Pred Value              0.991      0.879      1.0000       0.8750
## Neg Pred Value              0.953      0.992      0.9870       0.9932
## Prevalence                  0.731      0.192      0.0256       0.0513
## Detection Rate              0.718      0.186      0.0128       0.0449
## Detection Prevalence        0.724      0.212      0.0128       0.0513
## Balanced Accuracy           0.979      0.967      0.7500       0.9341
```

```
acc <- cM$overall['Accuracy']
F1unacc <- cM$byClass[1,'F1']
F1acc <- cM$byClass[2,'F1']
F1good <- cM$byClass[3,'F1']
F1vgood <- cM$byClass[4,'F1']


accuracy <- bind_rows(accuracy,
                      data_frame(method="svmRadial",
                                 accuracy = acc,
                                 F1unacc = F1unacc, F1acc = F1acc,
                                 F1good = F1good, F1vgood = F1vgood))
accuracy %>% knitr::kable()
```

method	accuracy	F1unacc	F1acc	F1good	F1vgood
knn	0.8782	0.9339	0.6531	0.5714	0.8571
lda	0.8974	0.9604	0.7692	0.2857	0.7692
naive_bayes	0.7308	0.8444	NA	NA	NA

method	accuracy	F1unacc	F1acc	F1good	F1vgood
svmLinear	0.9423	0.9649	0.8475	0.8889	1.0000
treebag	0.9231	0.9617	0.7925	0.8000	0.8571
gamLoess	0.8654	0.9607	0.7042	NA	NA
gam	0.8654	0.9607	0.7042	NA	NA
svmRadial	0.9615	0.9868	0.9206	0.6667	0.8750

We find that the svmRadial method gives an overall accuracy of 96.15%, with the F1 values of unacceptable, acceptable, good and very good classes as 0.9868, 0.9206, 0.6667, and 0.8750, respectively.

Next we train the hdda, or the high dimensional discriminant analysis algorithm:

```
################### hdda ###################
set.seed(1)
fit <- train(class ~ ., method = "hdda", data = train)
pred <- predict(fit, newdata = test)
cM <- confusionMatrix(pred, as.factor(test$class))
cM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction unacc acc good vgood
##      unacc   109   0    0     0
##      acc       5  28    0     1
```

```
##      good      0   2    4     0
##      vgood     0   0    0     7
##
## Overall Statistics
##
##                Accuracy : 0.949
##                  95% CI : (0.901, 0.978)
##     No Information Rate : 0.731
##     P-Value [Acc > NIR] : 1.61e-12
##
##                   Kappa : 0.885
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: unacc Class: acc Class: good Class: vgood
## Sensitivity                 0.956      0.933      1.0000       0.8750
## Specificity                 1.000      0.952      0.9868       1.0000
## Pos Pred Value              1.000      0.824      0.6667       1.0000
## Neg Pred Value              0.894      0.984      1.0000       0.9933
## Prevalence                  0.731      0.192      0.0256       0.0513
## Detection Rate              0.699      0.179      0.0256       0.0449
## Detection Prevalence        0.699      0.218      0.0385       0.0449
## Balanced Accuracy           0.978      0.943      0.9934       0.9375
```

```
acc <- cM$overall['Accuracy']
F1unacc <- cM$byClass[1,'F1']
F1acc <- cM$byClass[2,'F1']
F1good <- cM$byClass[3,'F1']
F1vgood <- cM$byClass[4,'F1']

accuracy <- bind_rows(accuracy,
                      data_frame(method="hdda",
                                 accuracy = acc,
                                 F1unacc = F1unacc, F1acc = F1acc,
                                 F1good = F1good, F1vgood = F1vgood))
accuracy %>% knitr::kable()
```

method	accuracy	F1unacc	F1acc	F1good	F1vgood
knn	0.8782	0.9339	0.6531	0.5714	0.8571
lda	0.8974	0.9604	0.7692	0.2857	0.7692
naive_bayes	0.7308	0.8444	NA	NA	NA
svmLinear	0.9423	0.9649	0.8475	0.8889	1.0000
treebag	0.9231	0.9617	0.7925	0.8000	0.8571
gamLoess	0.8654	0.9607	0.7042	NA	NA
gam	0.8654	0.9607	0.7042	NA	NA
svmRadial	0.9615	0.9868	0.9206	0.6667	0.8750
hdda	0.9487	0.9776	0.8750	0.8000	0.9333

This method performs discriminant analysis in high dimensions, and uses this infor-

mation for classification purposes.

We find that the hdda method gives an overall accuracy of 94.87%, with the F1 values of unacceptable, acceptable, good and very good classes as 0.9776, 0.8750, 0.8000, and 0.9333, respectively.

Next we train the C5.0Tree algorithm:

```
################### C5.0Tree ###################
set.seed(1)
fit <- train(class ~ ., method = "C5.0Tree", data = train)
pred <- predict(fit, newdata = test)
cM <- confusionMatrix(pred, as.factor(test$class))
cM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction unacc acc good vgood
##      unacc   113   5    0     0
##      acc       1  24    0     1
##      good      0   1    3     1
##      vgood     0   0    1     6
##
## Overall Statistics
##
##                Accuracy : 0.936
##                  95% CI : (0.885, 0.969)
```

```
##      No Information Rate : 0.731
##      P-Value [Acc > NIR] : 5.54e-11
##
##                   Kappa : 0.844
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: unacc Class: acc Class: good Class: vgood
## Sensitivity                 0.991      0.800      0.7500       0.7500
## Specificity                 0.881      0.984      0.9868       0.9932
## Pos Pred Value              0.958      0.923      0.6000       0.8571
## Neg Pred Value              0.974      0.954      0.9934       0.9866
## Prevalence                  0.731      0.192      0.0256       0.0513
## Detection Rate              0.724      0.154      0.0192       0.0385
## Detection Prevalence        0.756      0.167      0.0321       0.0449
## Balanced Accuracy           0.936      0.892      0.8684       0.8716
```

```r
acc <- cM$overall['Accuracy']
F1unacc <- cM$byClass[1,'F1']
F1acc <- cM$byClass[2,'F1']
F1good <- cM$byClass[3,'F1']
F1vgood <- cM$byClass[4,'F1']
```

```
accuracy <- bind_rows(accuracy,
                      data_frame(method="C5.0Tree",
                                 accuracy = acc,
                                 F1unacc = F1unacc, F1acc = F1acc,
                                 F1good = F1good, F1vgood = F1vgood))
accuracy %>% knitr::kable()
```

method	accuracy	F1unacc	F1acc	F1good	F1vgood
knn	0.8782	0.9339	0.6531	0.5714	0.8571
lda	0.8974	0.9604	0.7692	0.2857	0.7692
naive_bayes	0.7308	0.8444	NA	NA	NA
svmLinear	0.9423	0.9649	0.8475	0.8889	1.0000
treebag	0.9231	0.9617	0.7925	0.8000	0.8571
gamLoess	0.8654	0.9607	0.7042	NA	NA
gam	0.8654	0.9607	0.7042	NA	NA
svmRadial	0.9615	0.9868	0.9206	0.6667	0.8750
hdda	0.9487	0.9776	0.8750	0.8000	0.9333
C5.0Tree	0.9359	0.9741	0.8571	0.6667	0.8000

This method uses the concept of entropy for classification.

We find that the C5.0Tree method gives an overall accuracy of 93.59%, with the F1 values of unacceptable, acceptable, good and very good classes as 0.9741, 0.8571, 0.6667, and 0.8000, respectively.

Next we train the rf, or the random forest algorithm:

```r
################### rf ###################
set.seed(1)
fit <- train(class ~ ., method = "rf", data = train)
pred <- predict(fit, newdata = test)
cM <- confusionMatrix(pred, as.factor(test$class))
cM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction unacc acc good vgood
##      unacc   113   7    0     0
##      acc       1  22    0     1
##      good      0   1    4     0
##      vgood     0   0    0     7
##
## Overall Statistics
##
##                Accuracy : 0.936
##                  95% CI : (0.885, 0.969)
##     No Information Rate : 0.731
##     P-Value [Acc > NIR] : 5.54e-11
##
##                   Kappa : 0.842
##
```

```
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: unacc Class: acc Class: good Class: vgood
## Sensitivity                 0.991      0.733      1.0000       0.8750
## Specificity                 0.833      0.984      0.9934       1.0000
## Pos Pred Value              0.942      0.917      0.8000       1.0000
## Neg Pred Value              0.972      0.939      1.0000       0.9933
## Prevalence                  0.731      0.192      0.0256       0.0513
## Detection Rate              0.724      0.141      0.0256       0.0449
## Detection Prevalence        0.769      0.154      0.0321       0.0449
## Balanced Accuracy           0.912      0.859      0.9967       0.9375
```

```
acc <- cM$overall['Accuracy']
F1unacc <- cM$byClass[1,'F1']
F1acc <- cM$byClass[2,'F1']
F1good <- cM$byClass[3,'F1']
F1vgood <- cM$byClass[4,'F1']

accuracy <- bind_rows(accuracy,
                      data_frame(method="rf",
                                 accuracy = acc,
                                 F1unacc = F1unacc, F1acc = F1acc,
                                 F1good = F1good, F1vgood = F1vgood))
```

```
accuracy %>% knitr::kable()
```

method	accuracy	F1unacc	F1acc	F1good	F1vgood
knn	0.8782	0.9339	0.6531	0.5714	0.8571
lda	0.8974	0.9604	0.7692	0.2857	0.7692
naive_bayes	0.7308	0.8444	NA	NA	NA
svmLinear	0.9423	0.9649	0.8475	0.8889	1.0000
treebag	0.9231	0.9617	0.7925	0.8000	0.8571
gamLoess	0.8654	0.9607	0.7042	NA	NA
gam	0.8654	0.9607	0.7042	NA	NA
svmRadial	0.9615	0.9868	0.9206	0.6667	0.8750
hdda	0.9487	0.9776	0.8750	0.8000	0.9333
C5.0Tree	0.9359	0.9741	0.8571	0.6667	0.8000
rf	0.9359	0.9658	0.8148	0.8889	0.9333

This method creates several decision trees on various subsets of the given dataset and takes their average to improve the predictive accuracy on the dataset. This information is then used for classification.

We find that the rf method gives an overall accuracy of 93.59%, with the F1 values of unacceptable, acceptable, good and very good classes as 0.9658, 0.8148, 0.8889, and 0.9333, respectively.

We check the accuracies of various algorithms in predicting the test dataset, and find that the svmRadial method gives the highest accuracy, with better F1 accuracies in the largest (unacc) and smallest (vgood) classes than the next contender. Thus, this

method is selected. We test its validity on the validation dataset.

Validation on validation dataset

```
################### Validation ###################
# We train the svmRadial method on the
# complete working dataset to improve its accuracy.
# The trained algorithm is then used to predict the
# car classes in the validation dataset by considering the
# vehicle characteristics. This is then compared to the
# actual values in the validation dataset to compute
# the accuracy and the confusion matrix of the algorithm.
################### svmRadial ###################
set.seed(1)
fit <- train(class ~ ., method = "svmRadial", data = working)
pred <- predict(fit, newdata = validation)
confusionMatrix(pred, as.factor(validation$class))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction unacc acc good vgood
##      unacc   123   0    0     0
##      acc       5  36    0     1
##      good      1   0    4     0
##      vgood     0   0    1     5
```

```
##
## Overall Statistics
##
##                Accuracy : 0.955
##                  95% CI : (0.912, 0.98)
##     No Information Rate : 0.733
##     P-Value [Acc > NIR] : 1.24e-14
##
##                   Kappa : 0.896
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: unacc Class: acc Class: good Class: vgood
## Sensitivity                 0.953      1.000      0.8000       0.8333
## Specificity                 1.000      0.957      0.9942       0.9941
## Pos Pred Value              1.000      0.857      0.8000       0.8333
## Neg Pred Value              0.887      1.000      0.9942       0.9941
## Prevalence                  0.733      0.205      0.0284       0.0341
## Detection Rate              0.699      0.205      0.0227       0.0284
## Detection Prevalence        0.699      0.239      0.0284       0.0341
## Balanced Accuracy           0.977      0.979      0.8971       0.9137
```

```
################### End of validation ###################
```

Thus, we get 95.5% accuracy and a p-value of 1.24e-14 in this method. The confusion matrix shows that most of the cars have been correctly classified, and any deviations have only been to the nearby classes. In this way, the machine learning algorithm can easily replicate what a human would have done. Quite an excellent result for a few lines of code!

Conclusion

Summary

This project aimed to create a machine learning procedure to classify cars based on their attributes. To do that, we worked with UCI's machine learning repository dataset, which was downloaded, cleaned and inspected.

The downloaded dataset was then partitioned into working and validation datasets. The working dataset was further subpartitioned into train and test datasets. Several machine learning algorithms were trained on the train dataset and tested on the test dataset, and their accuracies and other parameters of the confusion matrix were analysed. Based on these, the svmRadial algorithm was selected as the best algorithm.

This algorithm was then validated on the validation dataset. To do that, the algorithm was re-trained on the complete working dataset, and then was used to predict the car classifications based on the validation dataset attributes. We found that the trained algorithm achieved an accuracy of over 95%, and a perusal of the confusion matrix revealed that any errors made were close errors.

In this way, we found that machine learning algorithms can be used to automate the process of vehicle classification. The accuracy achieved was similar to, if not higher than, one that would have been achieved through manual methods. Given the rapidity and robustness of the machine learning methods, it can be concluded that they offer significant advantages over manual methods.

Potential impact

The project touches upon an area that is witnessing practical implementation. Several used good markets have come up in recent years, including, but not limited to giants

such as OLX, eBay, KEH, Adorama, and Amazon. When looking at their listings, we find the quality of the item offered for sale, in such terms as open box, demo, excellent plus, excellent, excellent minus, very good, good, fair, and parts. While noting the objective attributes of the items offered for sale is an easy task, their categorisation into one of these categories by humans is not only time and labour intensive, but also leaves the possibility of subjectivity - that many assessers will differ in their ratings. In such cases, machine learning algorithms provide a fast, cost-effective, reliable and robust alternative.

We believe that the near future will witness several implementations of machine learning algorithms to achieve such objectives.

Limitations and future work

Being a demonstration project, we went with the default parameters for training and testing machine learning algorithms. During deployment in actual situations, probably more time will be spent on analysing more algorithms and tuning their parameters to achieve a greater level of accuracy and robustness. In particular, the empirical observations regarding certain single parameters (such as safety class) being able to give final classifications can be explicitly incorporated in the machine learning models.

In this project, we looked at a small number of attributes that were available in the dataset, but we realise that companies may wish to make use of many more number of attributes. Thus, implementation of machine learning in practical situations would require gathering of all the attributes being used by the companies in the field and training and testing the machine learning algorithms on them.

In the future, we would like to incorporate these points and work on actual implementation of machine learning for one of the firms working in this field. In particular, it would be interesting to note how the machine learning methods fare in comparison to the business as usual, manual methods.