2. Based on the following table, we'll construct a decision tree using ID3 algorithm.

	Outlook	Temperature	Humidity	Windiness	Play
1	Sunny	Hot	High	FALSE	No
2	Sunny	Hot	High	TRUE	No
3	Overcast	Hot	High	FALSE	Yes
4	Rainy	Mild	High	FALSE	Yes
5	Rainy	Cool	Normal	FALSE	Yes
6	Rainy	Cool	Normal	TRUE	No
7	Overcast	Cool	Normal	TRUE	Yes
8	Sunny	Mild	High	FALSE	No
9	Sunny	Cool	Normal	FALSE	Yes
10	Rainy	Mild	Normal	FALSE	Yes
11	Sunny	Mild	Normal	TRUE	Yes
12	Overcast	Mild	High	TRUE	Yes
13	Overcast	Hot	Normal	FALSE	Yes
14	Rainy	Mild	High	TRUE	No
15	Overcast	Cool	High	FALSE	No

Let's calculate the entropy of the table:

$$P(\text{No}) = \frac{6}{15}\,, \quad P(\text{Yes}) = \frac{9}{15} \quad \Rightarrow \quad H(\text{Initial}) = -\left[P(\text{No})\log_2 P(\text{No}) + P(\text{Yes})\log_2 P(\text{Yes})\right] = \\ = -\frac{6}{15}\log_2\frac{6}{15} - \frac{9}{15}\log_2\frac{9}{15} = 0.971\,.$$

If we are going to split the table, we need to calculate entropy (and entropy gain) for all attributes per following formula:

$$H(A) = \sum_{A_i \in A} P(A_i)H(A_i) = -\sum_{A_i \in A} P(A_i) \sum_{C_j \in C} P(C_j|A_i)\log_2 P(C_j|A_i)\,,$$

where A is the chosen attribute, A_i its possible values, C is the class with values C_j and $H(A_i)$ the entropy for branch $A = A_i$. Now let's do it so for all attributes:

- Outlook

$$P(\text{Sunny}) = \frac{5}{15}\,, \quad P(\text{Overcast}) = \frac{5}{15}\,, \quad P(\text{Rainy}) = \frac{5}{15}\,,$$
$$P(\text{No}|\text{Sunny}) = \frac{3}{5}\,, \quad P(\text{Yes}|\text{Sunny}) = \frac{2}{5}\,, \quad P(\text{No}|\text{Overcast}) = \frac{1}{5}\,, \quad P(\text{Yes}|\text{Overcast}) = \frac{4}{5}\,,$$
$$P(\text{No}|\text{Rainy}) = \frac{2}{5}\,, \quad P(\text{Yes}|\text{Rainy}) = \frac{3}{5}\,,$$

$$H(\text{Outlook}) = P(\text{Sunny})H(\text{Sunny}) + P(\text{Overcast})H(\text{Overcast}) + P(\text{Rainy})H(\text{Rainy}) = \\ = \frac{1}{3}(0.971 + 0.722 + 0.971) = 0.888\,.$$

Thus, the information gain is $IG(\text{Outlook}) = H(\text{Initial}) - H(\text{Outlook}) = 0.971 - 0.888 = 0.083$.

- Temperature

$$P(\text{Hot}) = \frac{4}{15}\,, \quad P(\text{Mild}) = \frac{6}{15}\,, \quad P(\text{Cool}) = \frac{5}{15}\,,$$
$$P(\text{No}|\text{Hot}) = \frac{2}{4}\,, \quad P(\text{Yes}|\text{Hot}) = \frac{2}{4}\,, \quad P(\text{No}|\text{Mild}) = \frac{2}{6}\,, \quad P(\text{Yes}|\text{Mild}) = \frac{4}{6}\,,$$
$$P(\text{No}|\text{Cool}) = \frac{2}{5}\,, \quad P(\text{Yes}|\text{Cool}) = \frac{3}{5}\,,$$

$$H(\text{Temparature}) = P(\text{Hot})H(\text{Hot}) + P(\text{Mild})H(\text{Mild}) + P(\text{Cool})H(\text{Cool}) =$$
$$= \frac{1}{15}(4 \cdot 1 + 6 \cdot 0.918 + 5 \cdot 0.971) = 0.958\,.$$

Thus, the information gain is $IG(\text{Temperature}) = H(\text{Initial}) - H(\text{Temperature}) = 0.971 - 0.958 = 0.013$.

- Humidity

$$P(\text{High}) = \frac{8}{15}\,, \quad P(\text{Normal}) = \frac{7}{15}\,, \quad P(\text{No}|\text{High}) = \frac{5}{8}\,, \quad P(\text{Yes}|\text{High}) = \frac{3}{8}\,,$$
$$P(\text{No}|\text{Normal}) = \frac{1}{7}\,, \quad P(\text{Yes}|\text{Normal}) = \frac{6}{7}\,,$$

$$H(\text{Humidity}) = P(\text{High})H(\text{High}) + P(\text{Normal})H(\text{Normal}) =$$
$$= \frac{1}{15}(8 \cdot 0.954 + 7 \cdot 0.918 + 7 \cdot 0.467) = 0.785\,.$$

Thus, the information gain is $IG(\text{Humidity}) = H(\text{Initial}) - H(\text{Humidity}) = 0.971 - 0.785 = 0.186$.

- Windiness

$$P(\text{Windy}) = \frac{6}{15}\,, \quad P(\text{Not windy}) = \frac{9}{15}\,, \quad P(\text{No}|\text{Windy}) = \frac{3}{6}\,, \quad P(\text{Yes}|\text{Windy}) = \frac{3}{6}\,,$$
$$P(\text{No}|\text{Not windy}) = \frac{3}{9}\,, \quad P(\text{Yes}|\text{Not windy}) = \frac{6}{9}\,,$$

$$H(\text{Windiness}) = P(\text{Windy})H(\text{Windy}) + P(\text{Not windy})H(\text{Not windy}) =$$
$$= \frac{1}{5}(2 \cdot 1 + 3 \cdot 0.918) = 0.951\,.$$

Thus, the information gain is $IG(\text{Windiness}) = H(\text{Initial}) - H(\text{Windiness}) = 0.971 - 0.951 = 0.020$.

Therefore, we split the tree at humidity, since the information gain is the largest here.

Next, we process the branch where humidity is high, the entropy of which is

$$H(\text{High humidity}) = -\frac{3}{8}\log_2\frac{3}{8} - \frac{5}{8}\log_2\frac{5}{8} = 0.954\,.$$

	Outlook	Temperature	Windiness	Play
1	Sunny	Hot	FALSE	No
2	Sunny	Hot	TRUE	No
3	Overcast	Hot	FALSE	Yes
4	Rainy	Mild	FALSE	Yes
8	Sunny	Mild	FALSE	No
12	Overcast	Mild	TRUE	Yes
14	Rainy	Mild	TRUE	No
15	Overcast	Cool	FALSE	No

- Outlook

$$P(\text{Sunny}) = \frac{3}{8}\,, \quad P(\text{Overcast}) = \frac{3}{8}\,, \quad P(\text{Rainy}) = \frac{2}{8}\,,$$
$$P(\text{No}|\text{Sunny}) = \frac{3}{3}\,, \quad P(\text{Yes}|\text{Sunny}) = \frac{0}{3}\,, \quad P(\text{No}|\text{Overcast}) = \frac{1}{3}\,, \quad P(\text{Yes}|\text{Overcast}) = \frac{2}{3}\,,$$
$$P(\text{No}|\text{Rainy}) = \frac{1}{2}\,, \quad P(\text{Yes}|\text{Rainy}) = \frac{1}{2}\,,$$

$$H(\text{Outlook}) = P(\text{Sunny})H(\text{Sunny}) + P(\text{Overcast})H(\text{Overcast}) + P(\text{Rainy})H(\text{Rainy}) =$$
$$= \frac{1}{8}(3\cdot 0 + 3\cdot 0.918 + 2\cdot 1) = 0.594\,.$$

Thus, the information gain is $IG(\text{Outlook}) = H(\text{High humidity}) - H(\text{Outlook}) = 0.954 - 0.594 = 0.360$.

- Temperature

$$P(\text{Hot}) = \frac{3}{8}\,, \quad P(\text{Mild}) = \frac{4}{8}\,, \quad P(\text{Cool}) = \frac{1}{8}\,,$$
$$P(\text{No}|\text{Hot}) = \frac{2}{3}\,, \quad P(\text{Yes}|\text{Hot}) = \frac{1}{3}\,, \quad P(\text{No}|\text{Mild}) = \frac{2}{4}\,, \quad P(\text{Yes}|\text{Mild}) = \frac{2}{4}\,,$$
$$P(\text{No}|\text{Cool}) = \frac{1}{1}\,, \quad P(\text{Yes}|\text{Cool}) = \frac{0}{1}\,,$$

$$H(\text{Temperature}) = P(\text{Hot})H(\text{Hot}) + P(\text{Mild})H(\text{Mild}) + P(\text{Cool})H(\text{Cool}) =$$
$$= \frac{1}{8}(3\cdot 0.918 + 4\cdot 1 + 1\cdot 0) = 0.844\,.$$

Thus, the information gain is $IG(\text{Temperature}) = H(\text{High humidity}) - H(\text{Temperature}) = 0.954 - 0.844 = 0.110$.

- Windiness

$$P(\text{Windy}) = \frac{3}{8}\,, \quad P(\text{Not windy}) = \frac{5}{8}\,, \quad P(\text{No}|\text{Windy}) = \frac{2}{3}\,, \quad P(\text{Yes}|\text{Windy}) = \frac{1}{3}\,,$$
$$P(\text{No}|\text{Not windy}) = \frac{3}{5}\,, \quad P(\text{Yes}|\text{Not windy}) = \frac{2}{5}\,,$$

$$H(\text{Temperature}) = P(\text{Windy})H(\text{Windy}) + P(\text{Not windy})H(\text{Not windy}) =$$
$$= \frac{1}{8}(3 \cdot 0.918 + 5 \cdot 0.971) = 0.951\,.$$

Thus, the information gain is $IG(\text{Temperature}) = H(\text{High humidity}) - H(\text{Temperature}) = 0.954 - 0.951 = 0.003$.

Thus, we split the high humidity branch at outlook, since its information gain is the largest. Let's continue with the same branch, which is split by outlook (following the depth-first or recursive approach of the algorithm). Since $P(\text{No}|\text{Sunny}) = 1$, it's clear that we don't need to process it separately: if the humidity is high and the it's sunny, we won't go and play tennis. However, if it's cloudy, the outcomes aren't that certain, yet:

	Temperature	Windiness	Play
3	Hot	FALSE	Yes
12	Mild	TRUE	Yes
15	Cool	FALSE	No

Without explicitly showing the calculations here, it is evident that if the temperature is cool, tennis is not played; otherwise, tennis is played. The resulting entropy after splitting by windiness is 2/3, whereas the entropy after splitting by temperature is 0.

The data in the last subbranch (high humidity and rainy) can be summarized in the following table

	Temperature	Windiness	Play
4	Mild	FALSE	Yes
14	Mild	TRUE	No

It's clear that temperature doesn't provide any information (gain); the final outcome is determined by windiness: if it's windy, then tennis is not played; if it's not windy, tennis is played.

Let's take the second branch (from root), i.e. normal humidity:

	Outlook	Temperature	Windiness	Play
5	Rainy	Cool	FALSE	Yes
6	Rainy	Cool	TRUE	No
7	Overcast	Cool	TRUE	Yes
9	Sunny	Cool	FALSE	Yes
10	Rainy	Mild	FALSE	Yes
11	Sunny	Mild	TRUE	Yes
13	Overcast	Hot	FALSE	Yes

The initial entropy is already quite low:

$$H(\text{Normal humidity}) = -\frac{1}{7}\log_2\frac{1}{7} - \frac{6}{7}\log_2\frac{6}{7} = 0.592\,.$$

- Outlook

$$P(\text{Sunny}) = \frac{2}{7}\,, \quad P(\text{Overcast}) = \frac{2}{7}\,, \quad P(\text{Rainy}) = \frac{3}{7}\,,$$
$$P(\text{No}|\text{Sunny}) = \frac{0}{2}\,, \quad P(\text{Yes}|\text{Sunny}) = \frac{2}{2}\,, \quad P(\text{No}|\text{Overcast}) = \frac{0}{2}\,, \quad P(\text{Yes}|\text{Overcast}) = \frac{2}{2}\,,$$
$$P(\text{No}|\text{Rainy}) = \frac{1}{3}\,, \quad P(\text{Yes}|\text{Rainy}) = \frac{2}{3}\,,$$

$$H(\text{Outlook}) = P(\text{Sunny})H(\text{Sunny}) + P(\text{Overcast})H(\text{Overcast}) + P(\text{Rainy})H(\text{Rainy}) =$$
$$= \frac{1}{7}(2\cdot 0 + 2\cdot 0 + 2\cdot 0.918) = 0.393\,.$$

Thus, the information gain is $IG(\text{Outlook}) = H(\text{Normal humidity}) - H(\text{Outlook}) = 0.592 - 0.393 = 0.199$.

- Temperature

$$P(\text{Hot}) = \frac{1}{7}\,, \quad P(\text{Mild}) = \frac{2}{7}\,, \quad P(\text{Cool}) = \frac{4}{7}\,,$$
$$P(\text{No}|\text{Hot}) = \frac{0}{1}\,, \quad P(\text{Yes}|\text{Hot}) = \frac{1}{1}\,, \quad P(\text{No}|\text{Mild}) = \frac{0}{2}\,, \quad P(\text{Yes}|\text{Mild}) = \frac{2}{2}\,,$$
$$P(\text{No}|\text{Cool}) = \frac{1}{4}\,, \quad P(\text{Yes}|\text{Cool}) = \frac{3}{4}\,,$$

$$H(\text{Temperature}) = P(\text{Hot})H(\text{Hot}) + P(\text{Mild})H(\text{Mild}) + P(\text{Cool})H(\text{Cool}) =$$
$$= \frac{1}{7}(1\cdot 0 + 2\cdot 0 + 4\cdot 0.811) = 0.464\,.$$

Thus, the information gain is $IG(\text{Temperature}) = H(\text{Normal humidity}) - H(\text{Temperature}) = 0.592 - 0.464 = 0.128$.

- Windiness

$$P(\text{Windy}) = \frac{3}{7}\,, \quad P(\text{Not windy}) = \frac{4}{7}\,, \quad P(\text{No}|\text{Windy}) = \frac{1}{3}\,, \quad P(\text{Yes}|\text{Windy}) = \frac{2}{3}\,,$$
$$P(\text{No}|\text{Not windy}) = \frac{0}{4}\,, \quad P(\text{Yes}|\text{Not windy}) = \frac{4}{4}\,,$$

$$H(\text{Windiness}) = P(\text{Windy})H(\text{Windy}) + P(\text{Not windy})H(\text{Not windy}) =$$
$$= \frac{1}{7}(3 \cdot 0.918 + 4 \cdot 0) = 0.393\,.$$

Thus, the information gain is $IG(\text{Windiness}) = H(\text{Normal humidity}) - H(\text{Windiness}) = 0.592 - 0.393 = 0.199$.

We see that both windiness and outlook produce the same information gain. To my knowledge, an arbitrary choice is made at this point. I opted for windiness.

The case where humidity is normal and weather not windy, tennis is played. However, when it's moderately humid but windy, additional split is needed:

	Outlook	Temperature	Play
6	Rainy	Cool	No
7	Overcast	Cool	Yes
11	Sunny	Mild	Yes

It's clear that temperature leaves us still undecided (corresponding entropy is $\frac{2}{3}$), whereas the outlook enables us to make the final decision (corresponding entropy is 0): if it's moderately humid, windy and rainy, tennis is not played; otherwise it is played.

The whole ordeal can be summarized in the following decision tree:

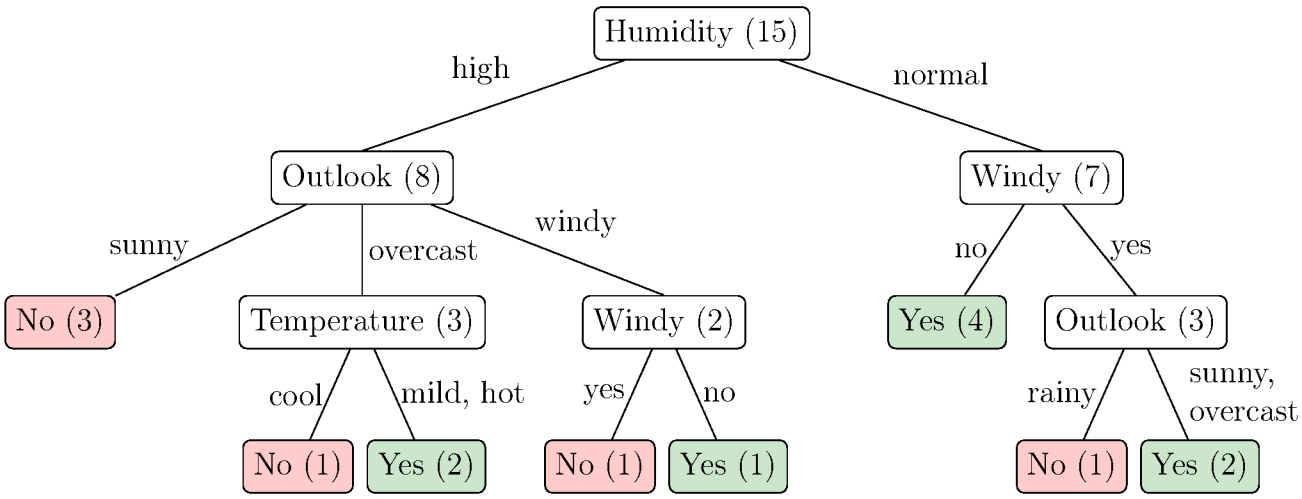

Figure 1: Decision tree awaiting testing. The numbers in parentheses are the number of observations the tree captured in the training phase.

If there is overcast, mild, high humidity and high wind weather, one should play tennis according to the decision tree: first we consider humidity (which is high, hence the left branch), then we consider the outlook (which is overcast, hence the middle branch) and the final decision is made based on the temperature (which is mild, hence „yes"); information about the wind wasn't used in answering the question.

3. We use RPART as our decision tree algorithm. The algorithm takes multiple parameters, from which I modified only

   - complexity parameter (CP) $\rightarrow$ 0.04 (default was 0.01);
   - maximum tree height $\rightarrow$ 10 (default was 30).

   Actually, the tree never reached height of 10, because CP stopped the tree growing earlier. In a nutshell, CP is the limit to the improvement of relative error (w.r.t misclassification error at the root node) if a node were to split: if the relative error reduced less than CP due to splitting, the node is not split[2]. The advantage of limiting the tree growth is that

   - the tree will not be overfitted;
   - readability and interpretability improves.

   The obvious disadvantage is that the leaf/terminal nodes aren't pure (i.e. it's a mixture of both classes). Here is the result:

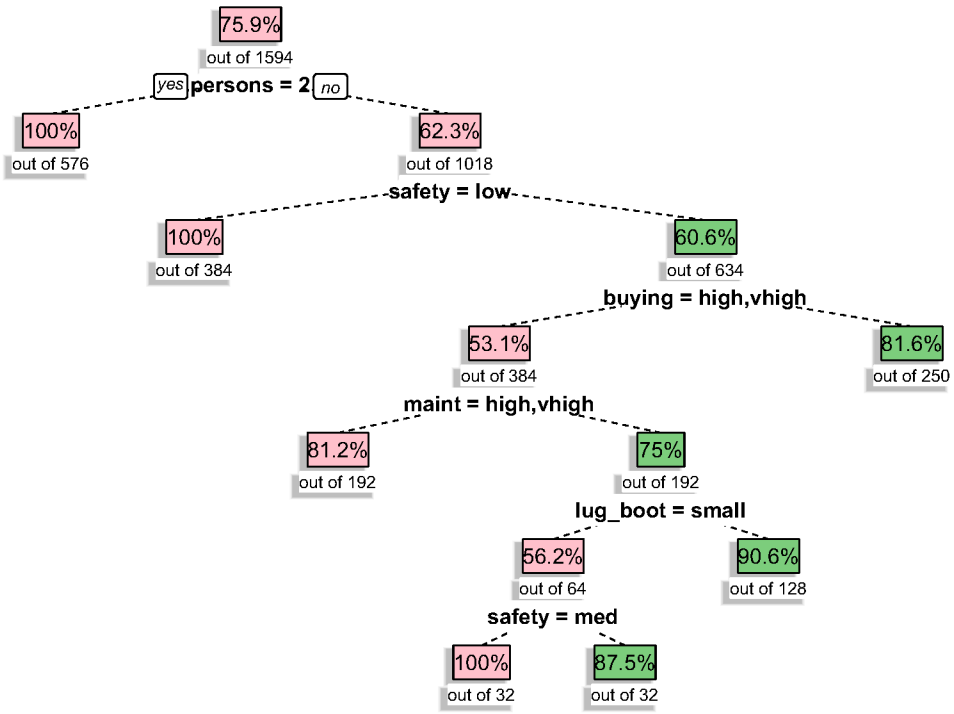

Figure 2: Decision tree for the `cars` dataset.

The tree should be interpreted in the following way:

- red node means *not accepted*, whereas green node means *accepted*;
- at the root node, 75.9% of all cars (1594) are not accepted;
- if only two persons fit in the car, then 100% of such cars are not accepted;
- otherwise, there are 1018 cars, 62.3% of which are not accepted etc etc;
- example terminal node: there are 250 cars, that accommodate more than two persons, safety is not low and the buying price is high or very high; the tree classified 81.8% of these cars as acceptable.

[2] http://stats.stackexchange.com/a/117909

The code that gave us tree fig. 2.

```
###################### RPART ############################

uri = "http://kodu.ut.ee/~ilyav/data/car.data"
cars.df = as.data.frame(read.csv(file = url(uri)))
dtree = function(df)
{
  cars.dtree = rpart::rpart(formula = class ~ ., data = df, method = "class",
                            control = rpart::rpart.control(cp = 0.04, maxdepth = 10))
  cars.dtree
}
cars.dtree = dtree(cars.df)

class_rate_tot = function(x, labs, digits, varlen) {
  paste0(signif(100*(1 - as.double(x$frame$dev) / x$frame$n),digits = 3), "%\n\nout of ", x$frame$n)
}
plot_tree = function(dtree)
{
  rpart.plot::prp(dtree,
                  box.col = c("palegreen3", "pink")[dtree$frame$yval],
                  digits = 1, type = 2, extra = 0, faclen = 0, varlen = 0,
                  node.fun = class_rate_tot, round = 0, xflip = T, branch.lty = 2,
                  shadow.col = "gray", compress = F)
}
plot_tree(cars.dtree)
cars.dtree.confu = table(predicted = predict(cars.dtree, type = "class"),
                         actual = cars.df$class)
```

Let's inspect the same data with association rule mining algorithm *apriori*, where we require the RHS of a rule to contain only `class` items, i.e. `acc` and `unacc`; minimum support and confidence is set to 0.2 and 0.5, respectively:

```
cars.apriori.rules = arules::apriori(
  data = cars.df,
  parameter = list(supp = 0.2, maxlen = 5, target = "rules", minlen = 2, conf = 0.5),
  control = list(verbose = F),
  appearance = list(default = "lhs", rhs = c("class=acc", "class=unacc"))
)
cars.apriori.rules = arules::sort(
  cars.apriori.rules,
  decreasing = T,
  by = "conf"
)
arules::inspect(cars.apriori.rules)
```

```
   lhs                 rhs           support   confidence lift
10 {safety=low}     => {class=unacc} 0.3613551 1.0000000  1.3173554
11 {persons=2}      => {class=unacc} 0.3613551 1.0000000  1.3173554
2  {maint=vhigh}    => {class=unacc} 0.2258469 0.8333333  1.0977961
3  {buying=vhigh}   => {class=unacc} 0.2258469 0.8333333  1.0977961
9  {lug_boot=small} => {class=unacc} 0.2823087 0.8108108  1.0681260
1  {doors=2}        => {class=unacc} 0.2045169 0.8009828  1.0551790
4  {buying=high}    => {class=unacc} 0.2032622 0.7500000  0.9880165
7  {lug_boot=med}   => {class=unacc} 0.2459222 0.7438330  0.9798924
6  {lug_boot=big}   => {class=unacc} 0.2308657 0.7187500  0.9468492
8  {safety=med}     => {class=unacc} 0.2239649 0.6648045  0.8757837
5  {persons=more}   => {class=unacc} 0.2020075 0.6338583  0.8350166
```

Interestingly, no strong rule doesn't contain `acc` item in the RHS, because the support of such rules is too low (the reason of which is that there are only 384 observations out of 1594 marked as such). By reducing the minimum support to 0.05 and limiting the RHS to `class=acc` only, we arrive at the following rules:

```
  lhs                           rhs         support    confidence lift
2 {persons=4,safety=high}    => {class=acc} 0.06775408 0.7500000  3.113281
1 {persons=more,safety=high} => {class=acc} 0.06022585 0.6620690  2.748276
3 {persons=more,safety=med}  => {class=acc} 0.05646173 0.5263158  2.184759
4 {persons=4,safety=med}     => {class=acc} 0.05646173 0.5172414  2.147091
```

By comparing these sets of association rules to the decision tree fig. 2, we see a quite good agreement:

- if the car accommodates only two persons, the car is not accepted;
- if the safety of the car is low, the car is not accepted;
- if the maintenance costs are very high, the car is not accepted; etc.

In other words, the attributes that are closer to the root node are also in the top of association rule table (ordered by confidence). However, the strongest rules contain only one item in the LHS, but the terminal nodes in the decision tree assume certain values for the attributes in their parent nodes (which is analogous to multiple items in the LHS of an association rule).

4. Let's add Naive Bayes and compare the confusion tables (using the same data that we trained the models with, i.e. all of data):

```
cars.nb = e1071::naiveBayes(formula = class ~ ., data = cars.df)
cars.nb.confu = table(predicted = predict(cars.nb, cars.df, type = "class"),
                      actual = cars.df$class)
```

Confusion tables:

		actual	
		accepted	unaccepted
predicted	accepted	348	62
	unaccepted	36	1148

Table 1: Confusion table for the decision tree (fig. 2; used the training data).

		actual	
		accepted	unaccepted
predicted	accepted	291	47
	unaccepted	93	1163

Table 2: Confusion table for Naive Bayes (used the training data).

Since our sample is quite small (only $\sim 1.5 \times 10^3$ records), there are more effective ways to estimate the quality of our models that reuse the original data, for instance k-fold cross-validation. In k-fold CV we

- first shuffle the data (reorder randomly);
- partition the data into $k = 10$ equal subsamples (called *folds*);
- loop over the folds ($k = 10$ iterations);
- in each iteration select the fold as our validation/testing data, the rest ($k - 1 = 9$ folds) are used as training dataset;
- calculate the confusion matrices for both Naive Bayes and RPART using the same fold;
- from these matrices, calculate precision, recall and accuracy;
- average the matrices and quality measures over $k = 10$ iterations.

CV is for validating how well our model performs (by model, I mean the algorithm with certain set of input parameters, i.e. Naive Bayes with default configuration, and RPART with CP equal to 0.04 and maximum height of 10). The following page shows the code that did exactly that.

```
acc = function(confu.tab)
{
  accuracy = sum(diag(confu.tab)) / sum(confu.tab)
  return(accuracy)
}
prec = function(confu.tab, class.lab)
{
  precision = confu.tab[class.lab, class.lab] / sum(confu.tab[class.lab,])
  return(precision)
}
recall = function(confu.tab, class.lab)
{
  tpr = confu.tab[class.lab, class.lab] / sum(confu.tab[,class.lab])
  return(tpr)
}
empty_confu = function()
{
  confu = as.table(matrix(0, 2, 2), levels = c("acc", "unacc"))
  colnames(confu) = c("acc", "unacc")
  rownames(confu) = c("acc", "unacc")
  return(confu)
}

kfold_cv = function(df, k, tree_training, outcome, outcome.lab)
{
  set.seed(123)
  shuffled = df[sample(nrow(df)),]
  folds = cut(seq(1, nrow(shuffled)), breaks = k, labels = F)

  tree.confu.sum = empty_confu()
  nb.confu.sum = empty_confu()

  tree.quality = list("accuracy" = 0, "precision" = 0, "recall" = 0)
  nb.quality = list("accuracy" = 0, "precision" = 0, "recall" = 0)

  for(i in 1:k)
  {
    test.idx = which(folds == i, arr.ind = T)
    test.data = shuffled[test.idx,]
    train.data = shuffled[-test.idx,]

    train.tree = tree_training(train.data)
    train.nb = e1071::naiveBayes(formula = as.formula(paste(outcome, "~ .")), data = train.data)

    tree.confu = table(predicted = predict(train.tree, test.data, type = "class"),
                       actual = test.data[[outcome]])
    nb.confu = table(predicted = predict(train.nb, test.data, type = "class"),
                     actual = test.data[[outcome]])

    tree.confu.sum = tree.confu.sum + tree.confu
    nb.confu.sum = nb.confu.sum + nb.confu

    tree.quality[["accuracy"]] = tree.quality[["accuracy"]] + acc(tree.confu)
    tree.quality[["precision"]] = tree.quality[["precision"]] + prec(tree.confu, outcome.lab)
    tree.quality[["recall"]] = tree.quality[["recall"]] + recall(tree.confu, outcome.lab)

    nb.quality[["accuracy"]] = nb.quality[["accuracy"]] + acc(nb.confu)
    nb.quality[["precision"]] = nb.quality[["precision"]] + prec(nb.confu, outcome.lab)
    nb.quality[["recall"]] = nb.quality[["recall"]] + recall(nb.confu, outcome.lab)
  }
  tree.confu.avg = round(tree.confu.sum / k)
  nb.confu.avg = round(nb.confu.sum / k)
  print(tree.confu.avg)
  print(nb.confu.avg)
  cat("quality\ttree\tnb\n")
  for(n in c("accuracy", "precision", "recall"))
    cat(paste(n, round(tree.quality[[n]] / k, digits = 3),
                 round(nb.quality[[n]] / k, digits = 3)), "\n")
}
kfold_cv(cars.df, 10, dtree, "class", "acc")
```

Here are the results:

		actual	
		accepted	unaccepted
predicted	accepted	35	7
	unaccepted	4	114

Table 3: Confusion table resulted from 10-fold CV of RPART model with $CP = 0.04$ and maximum height of 10.

		actual	
		accepted	unaccepted
predicted	accepted	28	5
	unaccepted	10	116

Table 4: Confusion table resulted from 10-fold CV of Naive Bayes model with input default parameters.

Quality measure	RPART	Naive Bayes
Accuracy	0.932	0.903
Precision	0.834	0.853
Recall	0.904	0.732

Table 5: Comparison of quality measures found by averaging over the quality measures of individual confusion tables found in each iteration of 10-fold CV.

From the last table, it looks like RPART is able to classify more data correctly (recall and accuracy is higher), but the misclassification rate withing class `acc` is higher. In other words, RPART classifies the records that were of class `acc` as such at a higher rate than Naive Bayes (higher recall for RPART), whereas the prediction of Naive Bayes contains fewer false positives (relatively speaking; slightly higher precision for Naive Bayes). So I'd say it's a tie here.

5. Here our target class or outcome (RHS in association rules) is the class `Survival`. Let's first build our models with full data and then test it with k-fold CV (or by splitting the data into two parts in case of *apriori*):

```
uri = "https://courses.cs.ut.ee/MTAT.03.183/2014_spring/uploads/Main/titanic.txt"
titanic.df = as.data.frame(read.csv(url(uri)))

titanic_rules = function(df, k)
{
  titanic.rules = arules::apriori(
    data = df, parameter = list(minlen = 2, supp = 0.02, conf = 0.5, target = "rules"),
    control = list(verbose = F),
    appearance = list(default = "lhs", rhs = c("Survived=No", "Survived=Yes"))
  )
  titanic.rules = arules::sort(titanic.rules, by = "conf", decreasing = T)
  return(head(titanic.rules, k))
}
titanic.rules = titanic_rules(titanic.df, 5)
arules::inspect(titanic.rules)

titanic_tree = function(df)
{
  titanic.dtree = rpart::rpart(formula = Survived ~ ., data = df, method = "class",
                               control = rpart::rpart.control(cp = 0.01, maxdepth = 10))
  titanic.dtree
}
titanic.dtree = titanic_tree(titanic.df)
plot_tree(titanic.dtree)
titanic.dtree.confu = table(predicted = predict(titanic.dtree, type = "class"),
                            actual = titanic.df$Survived)

titanic.nb = e1071::naiveBayes(formula = Survived ~ ., data = titanic.df)
titanic.nb.confu = table(predicted = predict(titanic.nb, titanic.df, type = "class"),
                         actual = titanic.df$Survived)
```

Here are the results:

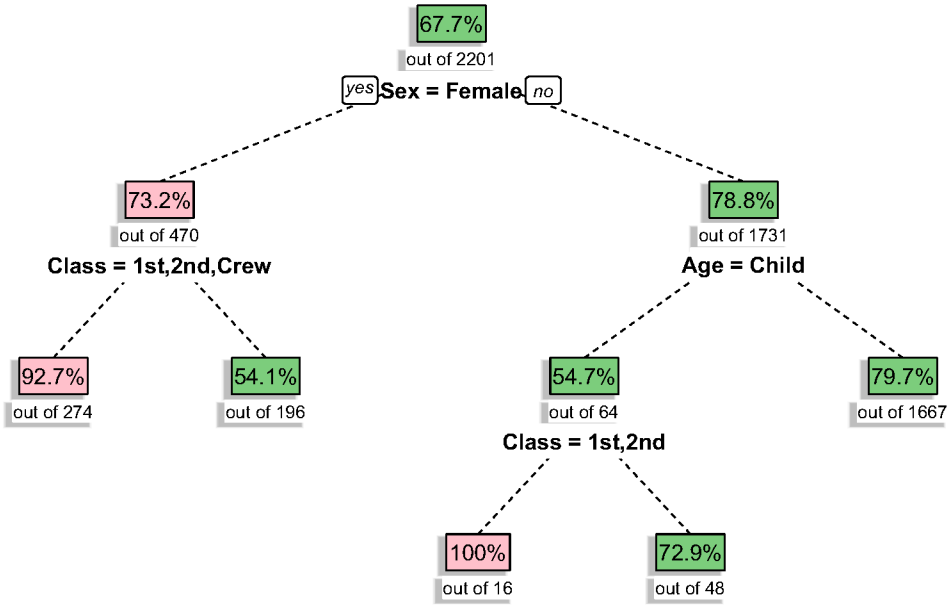

Figure 3: Decision tree for the `titanic` dataset. Complexity parameter is set to the default (0.01).

```
   lhs                                rhs            support    confidence lift
13 {Class=1st,Sex=Female}          => {Survived=Yes} 0.06406179 0.9724138  3.010243
25 {Class=1st,Sex=Female,Age=Adult} => {Survived=Yes} 0.06360745 0.9722222  3.009650
24 {Class=2nd,Sex=Male,Age=Adult}   => {Survived=No}  0.06996820 0.9166667  1.354083
10 {Class=2nd,Sex=Female}          => {Survived=Yes} 0.04225352 0.8773585  2.715986
11 {Class=2nd,Sex=Male}            => {Survived=No}  0.06996820 0.8603352  1.270871
```

Listing 1: Association rules found for `titanic` dataset. Minimum support and confidence is set to 0.02 and 0.5, respectively.

		actual	
		Didn't survive	Survived
predicted	Didn't survive	1470	441
	Survived	20	270

Table 6: Confusion table of RPART model with all `titanic` data used in the training.

		actual	
		Didn't survive	Survived
predicted	Didn't survive	1364	362
	Survived	126	349

Table 7: Confusion table of Naive Bayes model with all `titanic` data used in the training

By using the same 10-fold CV approach as we did in the previous exercise, we arrive at the following confusion tables and quality scores:

```
kfold_cv(titanic.df, 10, titanic_tree, "Survived", "Yes")
```

		actual	
		Didn't survive	Survived
predicted	Didn't survive	147	45
	Survived	2	26

Table 8: Confusion table resulted from 10-fold CV of RPART model for the `titanic` data.

		actual	
		Didn't survive	Survived
predicted	Didn't survive	136	36
	Survived	13	35

Table 9: Confusion table resulted from 10-fold CV of Naive Bayes model for the `titanic` data.

Quality measure	RPART	Naive Bayes
Accuracy	0.786	0.779
Precision	0.927	0.738
Recall	0.367	0.494

Table 10: Comparison of quality measures found by averaging over the quality measures of individual confusion tables found in each iteration of 10-fold CV (`titanic` data).

So far it looks like RPART is better than Naive Bayes, although the latter has better recall figure.
Here is quick testing of the association rules: I split the data into two parts (training 2/3 and testing 1/3); then counted the number of people that were expected to survive:

```
set.seed(123)
titanic.shuffled = titanic.df[sample(nrow(titanic.df)),]
titanic.training = head(titanic.shuffled, 2 * nrow(titanic.shuffled) / 3)
titanic.testing = tail(titanic.shuffled, nrow(titanic.shuffled) / 3)
titanic.rules = titanic_rules(titanic.training, 5)
arules::inspect(titanic.rules)
t1 = subset(titanic.testing, Class == "1st" & Sex == "Female")
t2 = subset(titanic.testing, Class == "1st" & Sex == "Female" & Age == "Adult")
t3 = subset(titanic.testing, Class == "2nd" & Sex == "Male" & Age == "Adult")
t4 = subset(titanic.testing, Class == "2nd" & Sex == "Female")
t5 = subset(titanic.testing, Class == "2nd" & Sex == "Female" & Age == "Adult")

cat(paste(nrow(subset(t1, Survived == "Yes")), nrow(subset(t1, Survived == "No"))))
cat(paste(nrow(subset(t2, Survived == "Yes")), nrow(subset(t2, Survived == "No"))))
cat(paste(nrow(subset(t3, Survived == "Yes")), nrow(subset(t3, Survived == "No"))))
cat(paste(nrow(subset(t4, Survived == "Yes")), nrow(subset(t4, Survived == "No"))))
cat(paste(nrow(subset(t5, Survived == "Yes")), nrow(subset(t5, Survived == "No"))))
```

```
   lhs                                rhs            support    confidence lift
13 {Class=1st,Sex=Female}          => {Survived=Yes} 0.06884799 0.9711538  2.949654
25 {Class=1st,Sex=Female,Age=Adult} => {Survived=Yes} 0.06816633 0.9708738  2.948803
24 {Class=2nd,Sex=Male,Age=Adult}   => {Survived=No}  0.06952965 0.9107143  1.357742
10 {Class=2nd,Sex=Female}          => {Survived=Yes} 0.04635310 0.8947368  2.717555
23 {Class=2nd,Sex=Female,Age=Adult} => {Survived=Yes} 0.04089980 0.8823529  2.679942
```

```
40 1
40 1
4 52
25 5
20 5
```

On the left hand side is the number of people that actually survived; on the right hand side is the number of people that didn't make it. By inspecting the above table and respective rules, it seems that the association rules work quite well, although the final numbers are really small (there were 734 records in the testing set; by applying any strong rule will give us relatively small final figures because their support is also small).

6. The model is overfitted if it's too complex for the amount of data. In other words: if the sample size the model is trained with is too sparse and the model itself too complex, the model will likely be overtrained (not robust; too specific to the data). One way to prune (i.e. remove too specific branches in decision trees) is to divide the sample intro three parts: one for training, the second for testing and tuning the tree and the last sample for testing and validation of the model. With the help of the second sample (after the training stage), errors are calculated and the tree is pruned in bottom up fashion.
   The second way to prune is to divide the dataset into two parts – one for training and the other for testing. After the training, each node is replaced with the most popular class it is likely to classify; if the accuracy and other quality scores will not change much, the change (i.e. the replacement) is kept.[3]
   Also, the complexity parameter also helps to choose optimal tree size (as described above). There really isn't a rule of thumb for the optimal tree size, because it depends on the problem (size of the sample, number of classes and attributes, the problem we want to solve etc).

[3] https://en.wikipedia.org/wiki/Pruning_(decision_trees)