Prof. Dr. Steffen Wagner
Angewandte Statistik
Fachbereich II
Berliner Hochschule für Technik

Exercise 10: Classification Trees and Bootstrapping

Machine Learning I – SoSe 24

This workshop covers hierarchical clustering and soft clustering. At the end of the worksheet there are a couple of written exercises for you to do at home, which should be good practice for the exam.

1 Preparations

1.1 RStudio Project

1. Open your Machine Learning 1 RStudio Project
2. Create an R Script file to perform this exercise

1.2 Required Packages

For this exercise you require the following additional R packages. Please make sure that you have installed them on your computer before coming to the workshop session for the case that Eduroam is not working.

```r
# check if packages can be loaded, i.e. they are already installed
library(ISLR2)       # contains data sets we might work with
library(rpart)       # package for tree fitting
library(rpart.plot)  # visualisation of tree structures
library(pROC)        # you should know what it is good for!
```

If you get an error at this stage, you need to install the packages.

2 Classification and regression trees (CART) with R

2.1 Brexit referendum results

In June 2016 the United Kingdom held a referendum to leave or remain in the European Union. The data have been published at a district level, the original data for each district is available from this website[1] but a user friendlier version is available in Moodle.

a) Download the data from Moodle, and load the data into R using the function `load()`. The data frame is called `Brexit`.

b) Spend a few minutes exploring the data, for example:

- How many districts (rows) are there?
- What are the variables in the data set?
- What proportion of voters voted leave and what proportion voted remain?
- What proportion of districts "voted leave'' i.e. had over 50% Leave in that district?\ Hint: `Status`
- Obtain a Boxplot for percentage of leave votes by Region.

c) Split the data into a training and a test data set with 300 observations in the training set.

d) Use the variables `Region`, `Electorate`, `Pct_Turnout`, `Votes_Cast` and `Pct_Rejected` to fit a **regression tree** to the outcome variable `Pct_Leave`.

- Fit and plot the default tree
- Fit and plot the full tree, use the functions `printcp()` and/or `plotcp()`, to find a good value of the complexity parameter for the pruned tree.
- Fit and plot the pruned tree.
- Find the mean squared error for the default tree and the pruned tree using the test data.

e) Fit a **classification tree** for the Leave `Status`.
Repeat the steps in parts (c-i) to (c-iii)

f) Obtain a confusion matrix for your test data. What are the name(s) of the district(s) which is/are predicted as *remain* but actually voted leave?
(Note this will depend on the seed you chose, when partitioning the data.)

g) Plot the ROC curve for the training data and obtain the AUC by adapting the commands below.

```r
require(???)

# predicted probability of "Leave"
predleavep <- predict(???, newdata=???, type = "prob")[,2]

# ROC analysis
roc.obj1 <- roc(???$Status, predleavep)
ggroc(???)
auc(???)
```

[1] https://www.electoralcommission.org.uk/find-information-by-subject/elections-and-referendums/past-elections-and-referendums/eu-referendum/electorate-and-count-information

2.2 Adjusting `rpart` parameters for unbalanced data

In previous weeks, you analysed a dataset with diabetes status (Yes/No) as the classification variable and age and body mass index (BMI) as predictor variables. The proportion of study participants with Diabetes is quite low.

This week you will look at fitting tree models to these data, including how to improve the performance of the tree model when the classification variable is very one sided.[2]

Classification tree with no adjustment

Load the `Diabetes` dataset, set the random seed, define a test set using 2000 participants and put the remaining observations into the training set.

```r
set.seed(50)
n <- dim(Diabetes)[1]
testidx <- sample(1:n, 2000)
test <- Diabetes[testidx, ]
train <- Diabetes[-testidx, ]
```

a) Use `rpart` to fit the default model when `Age` and `BMI` are used as predictor variables for the outcome `Diabetes`. Notice that this has no splits.

b) Fit the full tree using the complexity parameter as an argument `cp=0`.
This tree has 258 end-nodes, which is way too many. Obtain the cp-plot for this tree. Our pruning rule suggests the optimal tree is the second point, which corresponds to 11 end-nodes and a cp of 0.0046.

c) Fit this tree using `cp`$=$ 0.0046.
   - Plot the tree and obtain the ROC plot for the test data (see Exercise 2.1).
   - Obtain the AUC and calculate the sensitivity and specificity.
     Notice that the AUC is acceptable but the sensitivity is terrible at under 10%. This is often the case when the classes in the outcome variable are imbalanced.

Loss matrix

We will investigate changing the loss function matrix, which weights how critical a misclassification of a given type is. We will increase the (2,1) element of the matrix which corresponds to the false negatives being more heavily penalised.

```r
lossmat <- matrix(c(0, x, 1, 0), nrow = 2, ncol = 2)  # x will be set later, see below
lossmat
rptree <- rpart(YN ~ BMI + Age, data = train, parms = list(loss = lossmat))
```

d) Fit the model without the `cp` argument and starting with `x` equal to 1.

e) Gradually increase `x`, working through the sequence 1.1, 1.2, . . . , 2, 3, 4, ...
   - Each time plot the tree. If the tree changes obtain the AUC and the classification matrix.
   - Notice that as the value of x increases, the number of false negatives decreases, but with an increased number of false positives. This is the effect we are influencing via the loss matrix.

[2]NB: Both the following approaches, changing the loss matrix or the prior probabilities, are presented from a practical point of view, and the mathematical definitions are omitted. A quick explanation is that both methods result in a "generalised Gini index" or equivalent which is then used to quantify the quality of a split. Further details can be found in https://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf

Priors

Find the proportion of participants without diabetes. Set `piNo` to be this value and refit the starting tree using.

```
rptree <- rpart(YN ~ BMI + Age, data = train, parms = list(prior= c(piNo, 1 - piNo)))
```

These are the default prior values which are used in the algorithm to weight the Gini scores

f) Decrease `piNo` in steps of 0.1. Obtain the classification each time and notice that the sensitivity increases as the value of `piNo` decreases.

Finally make a graphical comparison of all 3 approaches, the pruned model, a model with a loss matrix without pruning, and a tree using priors without pruning.

g) Refit one appropriate model of each type saving the tree as an object with different names. For the loss matrix and prior, choose values that are not too extreme.
h) Graphical comparison: what type of visualisation would be appropriate?
i) Compare the 3 AUC values.

In summary, changing the loss matrix or the priors is useful to obtain a better “starting” tree when the classes are very unbalanced. Pruning should still be used to improve on that starting point.

3 Bootstrap sampling

3.1 Understanding bootstrap sampling

We will take the vector $1, 2, \ldots, n$ and investigate the properties of bootstrapping this vector.

a) Start with $n = 10$ so that we can easily observe what is going on.

```r
n <- 10
vec <- 1:n
bs_sample <- sample(vec, size = n, replace = TRUE)
bs_sample
sort(bs_sample)
table(bs_sample)
table(table(bs_sample))

# number of unique values in the bootstrapped sample
n_in_bag <- length(unique(bs_sample))
n_in_bag

# number of the out of bag values
n_ooB <- n - n_in_bag
```

The code the following:

- defines the sample size `n` and the vector to be bootstrapped,
- takes a sample of size `n` with replacement,
- outputs the raw bootstrap sample,
- sorts the output,
- creates a frequency table of the bootstrap sample,
- outputs how many values were sampled once, twice, three times etc.: `table(table())`,
- outputs how many "unique values" are in this bootstrapped sample?
- and calculates how many observations are "out of bag" OOB. Run the code, checking the output at each stage, and repeat a few times to understand what is going on.

b) The following code investigates the asymptotic behaviour for the proportion of out of bag samples.

```r
n <- 100
B <- 100

p_ooB <- n_ooB <- rep(NA, B)

for(i in 1:B){
  vec <- 1:n
  bs_sample <- sample(vec, size = n, replace = TRUE)

  # number of the out of bag values
  n_ooB[i] <- n - length(unique(bs_sample))
  # proportion of out of bag values so far.
  p_ooB[i] <- sum(n_ooB[1:i])/(i * n)

}

mean(n_ooB)/n
plot(p_ooB, type="l", ylim = c(0.3, 0.4))
abline(h = c(1/3, exp(-1)), col = c("blue", "red"), lty = 2)
```

- Replace the `???` in this file.
- Look at the proportion of out of bag samples and the plot of how this changes with iteration number.
- Increase the value of B to approximate the asymptotic proportion of out of bag observations.
- What seems to be the asymptotic proportion? Compare the proportion you are getting with the approximate proportion of a third and $1/e$ resp..

3.2 Bootstrapped estimates and intervalls

The following code, is a similar example to the 0.2-Quantile example in the lecture and is based on the following code. The original data comes from a very skewed distribution. There is a population variance for this distribution. You first estimate the population variance using the sample variance of the original data.

The aim is to understand how accurate this estimate of the population variance is. This is done using bootstrapping.

```r
n <- 75  # sample size
lambda <- ??? # choose a parameter value somewhere between 0.01 to 0.05
x <- rexp(n, lambda) # simulate the main data
hist(x, breaks = 10, main = "Histogram of main data", xlab = "x")

mean(x) # mean of the original sample
var(x)  # variance of the original sample

# take one resample (length 75 with replacement)
# and calculate the variance of the resample
resample <- sample(x, n, replace = T)
var(resample)

B <- 100 # 100 Bootstrap resamples
bsvar <- rep(NA, B)
for(i in 1:B){
  resample <- sample(x, n, replace = TRUE)
  bsvar[i] <- var(resample) # store the resampled mean for this iteration
}

hist(bsvar, breaks = 15,
     main = "Histogram of the bootstrapped variances", xlab = "var(x)")

sd(bsvar) # bootstrap estimate of the standard deviation of our estimator
quantile(bsvar, c(0.025, 0.975)) # 95% Conf-Int

#how do you get a 90% CI?

#add the Conf-Int to the histogram
abline(v = quantile(bsvar, c(0.025, 0.975)), col = 2)
truevar <- 1/lambda^2
abline(v = truevar, col = 4)
```

The value `truevar` and the blue line is the theoretical variance obtained using the theory arising from the Exponential (λ) distribution (https://en.wikipedia.org/wiki/Exponential_distribution).

4 Written Exercises

4.1 Classification Trees

A training data set has an outcome variable Y with $K{=}3$ classes labelled A, B and C. Two tree models are obtained each with just two leaf nodes. The frequencies for each class of Y are given in the following table for each node and both trees.
Note that the assignment to Nodes 1 and 2 are the same for Classes A and B, only the Class C assignment is different.

Outcome	Node 1	Node 2
A	104	81
B	111	23
C	27	42

Outcome	Node 1	Node 2
A	104	81
B	111	23
C	1	68

a) Calculate $\widehat{p}_{m,k}$ for $m = 1, 2$ (nodes) and $k = 1, 2, 3$ (classes). Do this for both trees.

b) Give the predicted class for each node.

c) Calculate for both trees the
   i) classification error rate E,
   ii) Gini coefficient G and
   iii) entropy D. Use $\log_e()$ for the entropy.

d) Which of the two trees is preferred using (i) the classification error rate, (ii) Gini coefficient and (iii) entropy.

Hint for parts c) and d) calculate all values to at least 4 decimal places.

Solution:

a) Tree 1:

$$\widehat{p}_{1,\bullet} = (0.4298, 0.4587, 0.1116)$$
$$\widehat{p}_{2,\bullet} = (0.5548, 0.1575, 0.2877)$$

Tree 2:

$$\widehat{p}_{1,\bullet} = (0.4815, 0.5139, 0.0046)$$
$$\widehat{p}_{2,\bullet} = (0.4709, 0.1337, 0.3953)$$

b) For both trees, Node 1 predicts class B and Node 2 predicts class A.

c) Tree 1: $E = 0.5052$, $G = 0.5895$, $D = 0.9694$
Tree 2: $E = 0.5052$, $G = 0.5484$, $D = 0.8393$

d) Smaller is better, so (i) the classification error rate is not in favor for one of the two trees, (ii) Gini and (iii) Entropy both prefer Tree 2.