Introduction to classification and regression trees, random forests and model-based recursive partitioning in R

Day 1: Single trees

Exercise 1: Impurity measures

We have a root node, in which a proportion of .7999 of the observations belong to class 1. One of the splitting candidates is a categorical variable with two levels. If this variable would be used to split the observations, a proportion of .565 would go left, and a proportion of .435 would go right. The proportion of class 1 observations in the left node would be 1.0. The proportion of class 1 observations in the right node would be .54.

Would this split improve purity according to the Gini index? According to the Shannon entropy? According to the classification error?

Exercise 2: Predicting carseat sales

The Carseats dataset from the package ISLR contains data of child car seat sales in 400 different stores:

```
data("Carseats", package = "ISLR")
summary(Carseats)
?ISLR::Carseats
```

Our aim is to predict car seat sales using the other ten predictor variables in the dataset. Note that we do not need to specify all ten predictor variables, `Sales ~ .` would instruct R to regress Sales on all the remaining variables in the dataset.

a) Randomly separate the dataset in 300 training and 100 test observations (don't forget to set the random seed!)

b) Fit and plot a CART tree using the `tree()` function.

c) Obtain the optimal complexity parameter value using the `cv.tree()` function and prune the tree using the `prune.tree()` function.

d) What seems to be the most important predictor of carseat sales?

e) Using the `predict()` function, generate predictions for the test observations. For both the pruned and the unpruned CART tree, calculate the correlation with the true car seat sales in the test data. Did pruning the CART tree indeed improve predictive accuracy?

Exercise 3: Illustrating variable selection bias

Using function `rnorm()`, generate three standard normally distributed (that is, $\mu = 0$ and $\sigma = 1$) variables `x1`, `x2` and `y`, each with 100 observations. Round the values of `x2` to 0 decimal places. Note that `x1` now has much more levels than `x2`. Fit two trees that aim to predict `y` from `x1` and `x2`: one using the `tree()` function from the `tree` package and one using the `ctree()` function from the `partykit` package.

Did the algorithms create spurious splits? If so, were `x1` and `x2` equally often selected for splitting, or was there a selection bias towards `x1`? Repeat the sampling and fitting process ten times. Do you see the same pattern every time?

Exercise 4: Revisiting carseat sales

a) Using the 'ctree()' function, fit a conditional inference tree to the training observations of the Carseats data from Exercise 2.

b) plot the tree. Is the ctree the same as the pruned CART tree from Exercise 2? Do the two trees agree on the most important predictor of carseat sales?

c) Using the `predict()` function, generate predictions for the test observations from. For the ctree, calculate the correlation with the true car seat sales in the test data. Does the ctree predict better than the (un)pruned CART trees?

Exercise 5: Predicting Glaucoma

Load the `Glaucoma` dataset from the package `TH.data`:

```
data("GlaucomaM", package = "TH.data")
?TH.data::GlaucomaM
```

The data set contains 62 continuous predictor variables. The response is the binary factor `GlaucomaM$Class`.

a) Fit a classification tree with the `ctree` function.

b) Plot the tree. Does it indicate main and/or interaction effects? Of which variables?

Exercise 6: Logistic Regression Based Recursive Partitioning

For this exercise, we'll use the Pima Indians Diabetes dataset. We will partition the dataset based on a logistic regression model, predicting the probability of having diabetes based on the outcome of a glucose tolerance test.

```
# install.packages("mlbench")  ## use only if package mlbench is not yet installed
data("PimaIndiansDiabetes", package = "mlbench")
?mlbench::PimaIndiansDiabetes
```

a) Use the function `glmtree()` from package `partykit` to fit a logistic regression based recursive partition. Regress the diabetes variable on the glucose variable and specify all other variables as potential partitioning variables. Add the argument `family = binomial` to specify a logistic regression model.

b) Plot and print the tree and interpret the results. Do the subgroups differ in terms of the association between the outcome of the glucose intolerance test and the probability of having diabetes? Or do the subgroups mostly differ in their overall probablity of having diabetes?

Day 2: Stability and tree ensembles

Exercise 1: Glaucoma data revisited (1)

a) Use the `stabletree()` function from the `stablelearner` package to assess stability of the tree you fitted yesterday. Do not forget to set the random seed, as the `stabletree()` function uses bootstrap samples to assess stability.

b) Is the first variable in the tree grown in Exercise 4 often selected? And how about the variables that appear further down the tree?

c) Check whether there is multicollinearity in the data: Could that explain that the variables appearing in the tree are not selected in each of the bootstrap samples?

Exercise 2: Bootstrapping vs. subsampling

Below, the code for generating one of the datasets I used in the presentation sheets is provided. Using this data and the `stabletree()` function from the `stablelearner` package, evaluate the difference between subsampling and bootstrap sampling in terms of the selection with which noise variables are selected:

```r
set.seed(12)
x1 <- rnorm(250, mean = 7, sd = 4)
x2 <- rnorm(250, mean = 14, sd = 4)
y_cont <- 8 + 5*(x1 > 7 & x2 > 13) + rnorm(250, 0, 1.5)
x3 <- rnorm(250)
ex_data <- data.frame(x1, x2, x3, y_cont)
```

a) Fit a ctree which predicts `y_cont` from `x1`, `x2` and `x3`.

b) Use the `stabletree()` function to assess the tree's stability using bootstrap sampling (the default).

c) Assess the tree's stability using subsamples instead of bootstrap samples. (Hint: Use `?stabletree` and `?bootstrap` to see how you can change the sampler used).

d) Compare the selection frequencies you obtained with both sampling methods.

Exercise 3: Glaucoma data revisited (2)

a) Fit a random forest using the `cforest` function from package `party`, with either bootstrap sampling with size N or subsampling with size $.632 \cdot N$.

b) Using the `predict()` function, compute the percentage of correctly classified observations — based on all trees and based only on those trees for which the observations are out-of-bag (by adding `OOB = FALSE` or `OOB = TRUE` to the function call) — and compare your results. Determine the predicted response class of each case and create a confusion matrix.

c) Compute and plot the standard permutation importance for the predictor variables.

d) Do the `stabletree()` function and 'cforest()' function give the same 3 most important variables?

Set random seeds everywhere in the code where random sampling is involved.

Exercise 4: Carseat sales revisited

Using the `cforest()` function from the `party` package, fit a random forest and a bagged ensemble to the 300 training observations. Using the `predict()` function, calculate the mean squared error (MSE) for:

a) The training observations
b) The OOB predictions for training observations (add `OOB = TRUE` to the function call)
c) The test observations (specify `newdata` argument)

$$MSE = \frac{1}{N}\sum_i (y_i - \hat{y}_i)^2$$

d) Are the OOB training error estimates indeed less optimistic / more realistic estimates of the test error?

Exercise 5: Boston housing data

Fit a random forest and bagged ensemble of CART trees to the Boston housing data. This dataset reports the median value (`medv`) of owner-occupied homes in 506 districts in the Boston area (USA), together with 13 potential predictor variables reporting sociodemographic and other characteristics of the districts:

```r
library("randomForest")
?randomForest
data("Boston", package = "MASS")
?MASS::Boston
```

Use the `randomForest()` function from the `randomForest` package to predict the median value of owner occupied home. Fit bth a random forest and a bagged ensemble. You could create the ensembles using a formula statement (i.e., `medv ~ .`), but if you supply arguments `x`, `y`, `xtest` and `ytest`, you can evaluate the test error for random forests consisting of 1 through the specified number of trees to grow. You will use these estimates to evaluate the effect of the number of trees on accuracy and to compare the OOB error estimates with the error estimated on test data. Therefore, first separate the data into a training and test set:

```
set.seed(1)
train <- sample(1:nrow(Boston), nrow(Boston)/4)
x <- Boston[train, -14]
y <- Boston[train, 14]
xtest <- Boston[-train, -14]
ytest <- Boston[-train, 14]
```

a) Set the `ntree` argument to 750, and create two ensembles: a bagged ensemble, using all 13 predictor variables for selecting each split, and a random forest using `sqrt(13)` for selecting each split.

b) Inspect the test error (which can be found in the `$test$mse` slot of the randomForest object) and the OOB error (which can be found in the `$mse` slot of the randomForest object) for both ensembles. Create a plot where the number of trees in the ensemble is on the x-axis, and the MSE is on the y-axis. Draw different lines for the OOB and test error of the bagged ensemble, as well as the OOB and test error of the random forest. (Hint: use functions `plot()` and `lines()` and the `type`, `lty` and `col` arguments)

c) After which tree size does the MSE start to stabilize?

d) Which of the two ensembles is most accurate on the test data?

e) Does the OOB error give a better estimate of test error for the random forest, or for the bagged ensemble? Can you explain the difference?

Exercise 6: Boston housing data (2)

Use the `gbm()` function from the package `gbm` to Fit a boosted ensemble on the Boston housing training data. Probably, you will first need to install the package `gbm`. Specify `distribution = "gaussian"`, as the outcome variable is continuous.

a) Use 5 or 10-fold cross validation on the training data to find the optimal value of the number of trees (`n.trees`). If you feel like it, also determine the optimal shrinkage parameter or learning rate by cross validation. Specify a sensible value for `interaction depth` (or, alternatively, also determine the optimal value by cross-validation).

b) Fit a boosted tree ensemble on the training data, using the optimal parameter values you found above.

c) Compare the test error of the boosted tree ensemble with that of the random forest and bagged ensemble you fitted above.